
DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

**TRABAJO FIN DE GRADO**

GRADO EN SISTEMAS DE COMUNICACIÓN

# EVALUACIÓN DE TÉCNICAS DE OPTIMIZACIÓN APLICABLES A ENTORNOS AMI

AUTOR: MARIO SANZ RODRIGO

TUTOR: JOSE IGNACIO MORENO NOVELLA

LEGANÉS, OCTUBRE DE 2016

TÍTULO: EVALUACIÓN DE TÉCNICAS DE OPTIMIZACIÓN APLICABLES A ENTORNOS AMI

AUTOR: MARIO SANZ RODRIGO

TUTOR: JOSE IGNACIO MORENO NOVELLA

TRIBUNAL

PRESIDENTE: CARLOS BOUSOÑO CALZON

SECRETARIO: MARÍA DEL CARMEN RAGA ARROYO

VOCAL: MARIO GÓMEZ DÍAZ

La defensa del presente Trabajo de Fin de Grado se realizó el día 10 de Octubre de 2016 en Leganés, en la Escuela Politécnica Superior de la universidad Carlos III de Madrid, siendo calificada por el siguiente tribunal:

PRESIDENTE SECRETARIO VOCAL

# Agradecimientos

En primer lugar agradecer a mi tutor José Ignacio por la oportunidad de desarrollar este proyecto dentro del equipo de investigación del departamento de telemática. Agradecer también a mis compañeros en dicho equipo, Gregorio López López y Miguel Seijo Simo por ayudarme durante el desarrollo del trabajo y resolver las dudas que me iban surgiendo, así como la genial convivencia durante todos estos meses.

A mi familia por haber estado siempre ahí durante todos estos años de carrera, aguantando los momentos de agobio, las fechas en las que no podía desplazarme a Burgos con ellos debido a los exámenes y compartiendo conmigo los momentos de alegría.

A todos los amigos tanto dentro como fuera de la universidad que han compartido conmigo buenos y malos momentos durante estos años.

Por último agradecer a de forma muy especial a mi novia Ana, ya que fue ella la que me motivó e insistió para comenzar esta andadura en la universidad hace ya más de cinco años, sin ella hoy no estaría donde estoy, muchísimas gracias.

## Resumen

En la actualidad el uso de las tecnologías de la información y la comunicación se están aplicando al sector eléctrico en lo que se conoce como Smart Grids. El presente proyecto se centra en los entornos AMI (Infraestructuras de medición avanzada) que se encuentran en la denominada “última milla” de las redes de distribución eléctrica.

La comunicación realizada en estos entornos se basa en el estándar PLC denominado PRIME. Debido a que los cables de electricidad no fueron diseñados originalmente para utilizarse como medio de comunicación es necesario el uso de herramientas que permitan obtener información sobre el funcionamiento de estas redes para poder detectar posibles problemas.

Una de estas herramientas es el analizador de trazas PrimeAnalytics, desarrollado por el equipo de investigación del departamento de telemática de la Carlos III dentro del ámbito del proyecto de investigación nacional OSIRIS (Optimización de la Supervisión Inteligente de la Red de Distribución), financiado por el Ministerio de Economía y Competitividad y liderado por Unión Fenosa Distribución.

Con este trabajo se pretende investigar, implementar y evaluar diferentes opciones para mejorar el rendimiento de la herramienta PrimeAnalytics, así como el uso de los recursos, centrándose en la paralelización de bases de datos y el uso de la computación en la nube.

## Palabras clave

Infraestructuras de Medición Avanzada (AMI): Comunicaciones PLC de Banda Estrecha; Django Web Framework; Docker; Citus; Base de Datos; PoweRline Intelligent Metering Evolution (PRIME); Python; Red Eléctrica Inteligente; Computación en la Nube; Analizador de Trazas; Cloud; Topología

## Abstract

Currently the use of Information and Communications technology are being applied to the electric sector in what is known as Smart Grids. This project focuses on the environment AMI (Advanced Metering Infrastructure) that are in the so-called “last mile” of the electrical distribution networks.

The communication performed in these environments is based on PLC standard, called PRIME. Because that power lines were not originally designed to be used as a means of communication, it is necessary the use tools to obtain information on the operation of these networks and detect potential problems.

One such tool is the PrimeAnalytics trace analyzer, developed by the research team of the department of telematics of the Carlos III within the scope of national research project OSIRIS (Optimization of Intelligent Monitoring Distribution Network) funded by Ministry of Economy and Competitiveness and led by Union Fenosa Distribution. This paper aims to investigate, implement and evaluate different options to improve the performance of the PrimeAnalytics tool and the use of resources, focusing on the parallelization of Databases and the use of cloud computing.

## Keywords

Advanced Metering Infraestructure (AMI); Narrowband-PLC (NB-PLC); Django Web Framework; Docker; Citus; Data Base; PoweRline Intelligent Metering Evolution (PRIME); Python; Smart Grids; Cloud Computing; Analizador de trazas; Cloud; Topology

# Índice general

# Índice de figuras

# Índice de tablas

# Acrónimos

| | |
|---|---|
| AMI | Advanced Meterting Infraestructure |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| BBDD | Bases de Datos |
| CFP | Contention Free Period |
| COSEM | COmpanion Specification for Energy Metering |
| CSMA/CA | Carrier Sense Multiple Access with Collision Avoidance |
| DLMS | Device Language Message Specification |
| DPSK | Differential Phase Shift Keying |
| DSO | Distribution System Operator |
| FEC | Forward Error Correction |
| GNU | General Public License |
| IEEE | Institute of Electrical and Electronics Engineers |
| ITU-T | Telecommunication Standardization Sector of the International Telecommunications Union |
| LLC | Logical Link Control |
| LNID | Local Node IDentifier |
| LSID | Local Switch IDentifier |
| ID | IDentifier |
| MAC | Medium Access Control |
| MTU | Maximum Transmission Unit |
| MTV | Model-Template-View |
| MVC | Modelo-Vista-Controlador |
| M2M | Machine-to-Machine |
| NB-PLC | Narrowband Power Line Communications |
| OFDM | Orthogonal Frequency Division Multiplexing |
| ORM | Object Relational Mapping |
| OSGP | Open Smart Grid Protocol |
| OSIRIS | Optimización de la Supervisión Inteligente de la Red de Distribución |
| PLC | Power Line Communications |
| PDU | Protocol Data Unit |
| PNPDU | Promotion Needed PDU |
| PRIME | PoweRline Intelligent Metering Evolution |
| SNR | Signal-to-Noise-Ratio |
| SSH | Secure SHell |
| SFTP | SSH File Transfer Protocol |
| TIC | Tecnologías de la Información y la Comunicación |

# Capítulo 1. Motivation and Goals

This chapter begins providing an overview of the PLC communication technology used in the electricity sector, focusing on the PRIME protocol. The technological challenges of this kind of networks make necessary the use of analysis tools. Due to the computational complexity of these tools and the large amount of data they use, a series of optimizations are proposed as the main contribution of this project. After contextualizing the project, this section finishes indicating the objectives set as well as a brief summary of the structure followed in this document.

## 1.1. Motivation

Nowadays, the use of information and communications technology (ICT) are present in many areas of our daily lives; they offer great strides that until a few years ago were not possible. One of these developments, which has gained strength in the recent years is the use of ICT in the electricity sector, largely by so-called Smart Grids. These intelligent networks are responsible for efficiently integrate all actors connected to them. Hence, they achieve and ensure an efficient and sustainable power grid with low losses and high levels of quality and safety. [1]

These networks represent a major change in the way the distribution and energy consumption is conceived. The integration of renewable energiesy is not only increasing, but also changing energy flows. Now users are not only consumers, but can also produce electricity in the same network. And because of this, it acquires a bidirectional character with Smart Meters as a key point.

Machine-Machine (M2M or Machine-to-Machine) communications take an important role within these networks, which encompass all communications between devices connected to them without human intervention [2]. One of its main purposes is the management of Smart Meters, which reduces latency, increases availability and reduces the costs of deployment and operation. From a technical point of view Power Line Communications (PLC) technology is very interesting for direct communication with Smart Meters in what is known as the last mile of Infrastructure Advanced Metering (AMI) due to the use of the existant electric infrastructure as communication medium.

The use of PLC technology is particularly interesting for electricity distributors (DSO or Distribution System Operator), as it doesn’t require the deployment of new infrastructures

for communications. Because power cables are not intended for its use as a communication medium, they offer a rather hostile environment from the point of view of data transmission, due to factors such as noise or impedance variation. Despite this, PLC offers many advantages to DSOs, where a series of standards for PLC technology Narrowband (NB-PLC or Narrow Band Power Line Communications) are being widely used for communication between Smart Meters and data concentrators in the current AMI deployments. [3]

Within the NB-PLC available options on the market, PowerLine Intelligent Metering Evolution (PRIME) technology and Meters and More [4], are the most mature and widely used, being PRIME, widely deployed in Spain, with international projection and used by large DSOs such as Iberdrola and Fenosa. PRIME has a large number of certified equipment and systems from different manufacturers that has allowed the deployment of more than 10 million Smart Meters PRIME by DSOs in Europe and worldwide (eg Brazil or Australia). Besides, it is estimated to be around 15 million Smart Meters PRIME deployed only in Spain, due to compliance with the IET / 290/2012 directive in 2018.

The PRIME protocol has been defined by the PRIME Alliance [4]. Its aim is to establish a standard among equipment and systems from different vendors, and the PHY and MAC layer have been accepted as a standard by ITU-T [5] in 2012. PRIME standard is not subject to intellectual property rights, so PRIME is an economical, efficient and barrier-free solution, benefiting utilities and users.

However, in addition to the benefits of NB-PLC networks in general, and PRIME in particular, they have a number of technological challenges that may raise doubts regarding its performance. That is why simulation and analysis tools are extremely important to plan, evaluate, detect problems and make decisions by minimizing risks, time, resources and costs [6].

Among the analyzers used for PRIME networks there is PrimeAnalytics. Developed within the national research project OSIRIS (Optimization of Intelligent Monitoring Distribution Network), this PRIME analysis tool aims to detect and diagnose problems in last-mile networks formed by data concentrators and Smart Meters

This tool focuses on the analysis of three types of files: PRIME trace .csv file, .xml file with the current topology and csv file with the evolution of the topology. Using the analyzer you can see the network in more detail and identify possible failures.

It is important to emphasize that this tool provides two different kinds of information. On the one hand we can obtain information of the traces (e.g. packet types, number of uplink packets, number of downlink packets) and other information concerning the PRIME network nodes (e.g. levels of network nodes, number of children nodes, etc.)

This kind of tools make use of both data query ands storage, and handle a vast ammount of information due to the number of files that are loaded (data of long time periods, different scenarios....). All this affects the time it takes to return the query result. It is therefore important to implement optimization methods such as cloud computing or database parallelization.

Nowadays, the model of cloud computing is taking more and more strength in the IT sector. One of the great advantages of its use is that its security measures often exceed those of traditional infrastructure. Apart from the security point of view, using it provides great expandability of the resources, which means flexibility, efficiency and a reduction of the costs. The usual operating model is based on the demand for our application or tool, so the resources can be dynamically adapted to fit the requirements, thus only paying for the resources we need.

It is not uncommon to see every day how more and more companies are migrating their platforms to the cloud, more and more users are attracted to their security, scalability and transformative effect on businesses. Cloud systems provide cost-effective solutions that save money on equipment, service contracts, personal and power.

Furthermore, parallelization ensures database system performance at an acceptable rate, even in the case of increasing the size of the same or in the case of increasing the number of queries, apart from increasing availability by using replicas.

These systems consist of a series of nodes, each one with its own resources such as the processor, memory and hard disk. The goal of these nodes is to process different shards of the database, avoiding to consult a full table, thereby achieving a great improvement in speed when performing queries.

The aim of this TFG is to research, implement and evaluate different options to improve application performance and use of resources focusing on the parallelization of databases and the use of cloud computing.

Its important to point out that this bachelor thesis has been developed within the scope of national research project OSIRIS (Optimization of Intelligent Monitoring Network Distribution) [7], funded by the Ministry of Economy and Competitiveness and led by Union Fenosa Distribution (third electricity distributor nationally).

## 1.2. Goals

The main objective of this TFG is to investigate, implement and evaluate different options to improve application performance PrimeAnalytics.

To achieve the ultimate goal, the follogin sub-objectives are set.

- Planning and designing stages in which the project will be divided.
- Study of the operation of the tool PrimeAnalytics.
- Study of the different Cloud Platforms available.
- Study of posible solutions of parallelization for the databases.
- Implementation and compatibility study parallelization of the solution in the analysis tool traces PrimeAnalytics.
- Validation of performance optimization solution base don parallelization in the local environment.
- Migration of the PrimeAnalytics tool to a cloud environment.
- Documenting the work done.

## 1.3. Memory structure

This document is organized according to the following structure:

- Chapter 1: Motivation and Goals.

This chapter begins provided an overview of the PLC communication technology used in the electricity sector, emphasizing the PRIME protocol. Due to the technological challenges it is necessary to use analysis tools. Due to operation of these tools and the large amount of data they use, a series of optimizations is proposed which they are the basis of this project. After contextualizing the project, is finished indicating the objectives set by the project as well as a brief summary of the structure followed in memory.

- Chapter 2: State of the art.

This chapter describes the state of the art of the all tecnologies used and related in this project. For this, four large blocks will be addressed. In the first block,PLC narrowband technologies used in Europe and known as NB-PLC are explained. In the second block we will talk about the PRIME protocol, after addressing the most important aspects of the protocol operation, necessary to understand the operation of the tool PrimeAnalytics. After, in the third block, the operation of the tool, as well as language and tools used for development is explained. It is an important point when performing optimization. Finally, in

the fourth block optimization techniques, the parallelization of databases and cloud computing.

- Chapter 3: Design.

In this chapter it is described the design process of the solution of parallelization and integration for the proper functioning in PrimeAnalytics based on the Framework Django and system PostgreSQL database for subsequent migration to the Cloud platform. To do this, explains the various tools needed for development, the various functional requirements, the limitations and problems encountered when performing the deployment. To provide a clear picture of the implementations, a series of figures are shown, illustrating the current operation of the tool and operation expected by applying optimization techniques chosen. Finally, the main technical and design decisions that have been taken towards the development of this proyect.

- Chapter 4: Development and Integration.

In this chapter detailed the implementation of the optimization techniques chosen following the principles and methodology established design in Chapter 3. Will begin explaining the steps taken to develop the parallelization of the database, the decisions made during development, the problems encountered and the various modifications to the analyzer. Then, due to the advantages offered by the parallelization, the implementation of a new functionality for the tool is detailed. Finally, the migration process to the cloud environment is explained.

- Chapter 5: Testing and Evaluation.

This chapter describes the tests performed to evaluate the design of implementations, it defined two blocks of tests, the first is in a local environment and the second in a cloud environment. A turn, in local environment tests have been performed two methods to get the results, the first connects directly to the database to emulate consultations and the second measures the response times Ajax requests analyzer. The goal is assess whether the deployments are able to reduce response times. In addition to the evidence presented in this chapter, various tests that ultimately were not included due to its extension were made.

- Chapter 6: Conclusions and Future Works.

This chapter summarizes the main conclusions after the execution of this TFG, the personal opinion and a number of lines of futures works that can begin.

- Chapter 7: Budget.

This chapter presents the planning and development phases of the TFG and the budget breakdown in the materials and human resources costs.

# Capítulo 2. Estado del arte

En este capítulo se analizará el estado del arte aplicable de las tecnologías relacionadas y utilizadas en este proyecto. Para ello se abordarán 4 grandes bloques. En el primero de ellos se revisaran las tecnologías PLC de banda estrecha conocidas como NB-PLC utilizadas en Europa, seguidamente se hablará sobre el protocolo PRIME, tras abordar los aspectos más importantes de dicho protocolo se explicará el funcionamiento de la herramienta de análisis de trazas PrimeAnalytics así como la necesidad de conocer tanto el lenguaje como las herramientas utilizadas para su desarrollo debido a que será necesario a la hora de realizar el desarrollo de las optimizaciones elegidas. Por último se explicarán las técnicas de optimización a evaluar en las que destacan dos subgrupos, la paralelización de bases de datos y la computación en la nube.

## 2.1. Tecnologías PLC de banda estrecha

### 2.1.1. Visión global

Actualmente las redes de comunicaciones M2M tienen un papel relevante en el ámbito de las Smart Grid puesto que son clave para realizar la monitorización y control prácticamente en tiempo real de una gran cantidad de elementos. Debido a esto se ha de cumplir con una serie de requisitos exigentes desde el punto de vista técnico (p.ej., baja latencia, alta disponibilidad) [8] y económico (bajos costes de despliegue y operación) [3].

Debido a estos requisitos, las tecnologías PLC son las más relevantes en este tipo de entornos ya que representan un compromiso entre ambas perspectivas. Una de las ventajas más evidentes de las tecnologías PLC es el considerable ahorro debido a que utiliza el cableado de baja tensión para realizar las comunicaciones, por lo que evita la necesidad de realizar nuevos despliegues con el elevado coste que esto supondría. Por el contrario, el cableado de baja tensión es un medio hostil desde el punto de vista de las comunicaciones dado que inicialmente no fue diseñado para la transmisión de datos sino para transmitir potencia, lo cual supone grandes retos para su implementación.

Diversos estudios [9, 10, 11, 12] y proyectos pilotos llevados a cabo por distribuidoras eléctricas [13] apuntan a que NB-PLC es la tecnología más apropiada especialmente en la parte conocida como última milla en despliegues AMI, como interfaz entre los

concentradores de datos ubicados generalmente en los centros de distribución de baja tensión y los Smart Meters.

A nivel Europeo existen diversos estándares NB-PLC actualmente utilizados en el mercado como podemos observar en la Figura 1:


*Figura 1: Mapa del despliegue de tecnologías NB-PLC en Europa [14]*

- **Meters and More** es una tecnología cuya especificación lidera el grupo Enel, aunque sus capas inferiores también están siendo estandarizadas por IEC/CENELEC (CLC TS 50568-4). Presenta sus mayores niveles de penetración en mercados donde opera el grupo ENEL, tales como Italia (el 100% de los contadores italianos usan esta tecnología actualmente) o España (aproximadamente la mitad del parque de contadores inteligentes españoles utilizarán esta tecnología en 2018). Funciona en la banda CENELEC-A. Utiliza una única portadora, destacando por su robustez, aunque alcanza tasas de transmisión de datos bajas (9,6 kbps).

- **Open Smart Grid Protocol** (OSGP) es una tecnología promovida por el fabricante norteamericano Echelon, aunque sus capas inferiores están también estandarizadas por el IEC (IEC 14908.1). Presenta sus mayores tasas de despliegue en los países nórdicos y Rusia. Funciona en la banda CENELEC-A. Se trata de una tecnología mono portadora que alcanza tasas de transmisión de datos de hasta 3,6 kbps.

- **CX1** es una tecnología promovida por Siemens y cuyas capas inferiores están siendo estandarizadas también por IEC/CENELEC (CLC TS 50590). Actualmente se está desplegando en Austria. Funciona en la banda CENELEC-A. Utiliza una modulación multiportadora adaptativa que le permite alcanzar tasas de hasta 64 kbps.

- **G3** es una tecnología cuya especificación lidera la distribuidora EDF y el fabricante de chipsets Maxim, aunque sus capas PHY y de enlace han sido publicadas como estándar por la ITU-T recientemente (ITU-T G9903). Presenta sus mayores niveles de penetración en mercados donde opera EDF, como Francia. Inicialmente definida para que funcionase en la banda CENELEC-A, ha sido extendida recientemente para poder ser utilizada en el mercado americano (FCC) y asiático (ARIB). Se trata de una tecnología multiportadora que alcanza tasas de hasta 34 kbps y que ha destacado desde sus inicios por ser especialmente robusta.

- Como ya se ha comentado, **PoweRline Intelligent Metering Evolution (PRIME)** es una tecnología promovida por la PRIME Alliance, liderada por distribuidoras eléctricas españolas como Iberdrola y Unión Fenosa Distribución y por fabricantes de chipsets como Texas Instruments. Sus capas PHY y de enlace también han sido publicadas como estándar por la ITU-T (ITU-T G.9904) [5]). Al igual que G3, fue inicialmente definida para que funcionase en la banda CENELEC-A y ha sido extendida recientemente para poder ser utilizada en el mercado americano (FCC) y asiático (ARIB). Es una tecnología multiportadora que puede llegar a alcanzar tasas de hasta 128,6 kbps (en su versión para la banda CENELEC-A). G.hnem [15] se trata de un esfuerzo de la ITU-T por homogeneizar G3 y PRIME, aunque su implementación es computacionalmente más compleja que las anteriores. Su utilización actualmente es baja.

Actualmente en España PRIME destaca como una tecnología NB-PLC prometedora dentro de las tecnologías disponibles debido a que se trata de un estándar PLC maduro, consolidado en el territorio nacional debido al uso que realizan de él grandes distribuidoras como Fenosa e Iberdrola, además de contar con una gran proyección internacional. PRIME cuenta con un gran número de equipos y sistemas certificados de diferentes fabricantes, que gracias a las distribuidoras han permitido el despliegue de más de 10 millones de Smart Meters PRIME.

Dado que el analizador PrimeAnalytics se centra en esta tecnología, y el objetivo de optimización, en el cual se basa este Trabajo de Fin de Grado está relacionado con la optimización de dicho analizador de trazas mediante el uso de la computación en la nube y la paralelización de bases de datos, el próximo apartado está dedicado a la tecnología PRIME, resumiendo brevemente sus principales detalles técnicos.

### 2.1.2. PRIME

El protocolo PRIME ha sido definido por la PRIME Alliance [4], cuyo objetivo es establecer un conjunto de estándares públicos. Cuenta con unas especificaciones exhaustivas y detalladas sin estar sujetos a derechos de propiedad intelectual, es decir, PRIME es una

solución económicamente eficiente y sin barreras que beneficia a las compañías eléctricas y a sus usuarios.

Es una tecnología NB-PLC de segunda generación donde su versión 1.3.6 de la especificación de las capas PHY, MAC y de Convergencia ha sido aceptada como estándar por la ITU-T desde 2012 [5]. La versión 1.4 expande el espectro de frecuencia utilizado para poder operar en los mercados americano y asiático e incluye algunas funciones para incrementar la robustez de la comunicación a nivel de la capa PHY y MAC.

Desde una visión de la capa PHY, PRIME opera en la banda de 41-89 kHz (v1.3.6) o de 3-500 kHz (v1.4), usando la modulación OFDM para hacer un uso más eficiente del espectro. A parte de la modulación ODFM, los dispositivos PRIME pueden usar modulaciones DPSK, DQPSK o D8PSK. Además, implementa el uso de FEC para poder recuperarse ante errores introducidos por el canal.

Desde la perspectiva de la capa física, la velocidad de transmisión en estas redes puede ir desde los 5.4 kbps hasta los 1028.8 kbps, dependiendo de la combinación de modulación digital y FEC utilizadas.

A continuación se va a explicar el funcionamiento básico del estándar PRIME con el fin de asentar los conceptos básicos de funcionamiento que serán útiles para entender el funcionamiento del analizador así como los requisitos y problemas a la hora de realizar la optimización.

Según el estándar PRIME, a nivel MAC, se definen dos tipos básicos de nodos: Nodo Base y nodo de Servicio. En la terminología AMI, el nodo Base es conocido como concentrador de datos y los nodos de Servicio suelen ser los Smart Meters, que pueden funcionar a su vez como Switches de los que cuelgan otros nodos de Servicio, creando así una topología en árbol que va desde un único concentrador como nodo raíz o nodo padre, hasta multitud de nodos hijos, formando una topología de varios niveles.

En lo referente al nodo de servicio, este permanece en un estado de desconexión, en dicho estado sólo puede recibir balizas en el canal y enviar PNPDUs (Promotion Needed PDU). Dichos nodos de Servicio pueden tener una serie de estados, Disconnected, Terminal y Switch, siendo una topología variable a lo largo del tiempo. Mediante una serie de procesos de registro/desregistro y promoción/degradación, un nodo de Servicio puede pasar de desconectado a Terminal, o a Switch. Más adelante se detallarán los procesos de registro - desregistro y promoción - degradación.

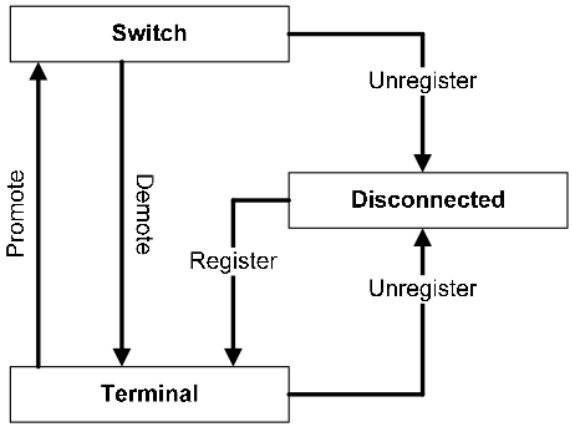



*Figura 2: Esquema de los tipos de Service Node*

Cada Service node debe mantener una Switch Table que es actualizada con la recepción de balizas desde cualquier nodo Switch nuevo en la red. El Service Node puede seleccionar cualquier nodo Switch contenido en la Switch Table e iniciar el proceso de registro contra ese Switch Node (el criterio para seleccionar el switch node a partir de la switch table no aparece específicamente detallado en la recomendación PRIME).

El Service node debe escuchar en el canal un determinado tiempo (macMinSwtichSearchTime) antes de decidir que no está recibiendo balizas. Si no se reciben balizas en ese tiempo establecido, el service node debe difundir el PNPDU para solicitar la promoción. El estándar fija que dicho service node podrá transmitir un determinado número máximo de PNPDUs (macMaxPromotionPdu) por unidad de tiempo (macPromotionPduTxPeriod).

En el caso del Base node o también denominado concentrador, ha de realizar las siguientes funciones para garantizar el correcto funcionamiento de la red.

- **Transmisión de balizas**: el Base node y todos los Switch node de la subred difundirán balizas en un intervalo de tiempo fijado por el concentrador. El Base node transmite siempre una baliza por trama. Los Switch node transmiten dichas balizas con la frecuencia predefinida por el Base node a la hora de realizar su promoción.

- **Promotion y Demotion de los terminales y switchs**: todas las peticiones de promoción generadas por los terminales tras la recepción de los PNPDUs se dirigen al Base node. El Base node mantiene una tabla con todos los Switch node de la subred y asigna un único identificador denominado SID (Switch ID) a las nuevas solicitudes entrantes. El proceso de demotion puede ser iniciado por el Base nodo o ser solicitada por el Switch node.

- **Gestión de los registros**: el Base node es el encargado de recibir todas las peticiones de registro por parte de los nuevos nodos que están intentando conectarse a la red. El

base node debe procesar cada petición de registro recibida respondiendo con un mensaje de aceptación o de rechazo. Una vez que Base node acepta el registro de un nuevo Service node, dicho nodo es identificado con un único NID (Node ID) que se usará cada vez que se realice una comunicación en la subred.

Aparte de las funciones ya mencionadas, el Base node se encarga de funciones como el arbitraje de acceso al canal, de distribución de secuencias random para la obtención de claves de encriptación y de la gestión de grupos de difusión.

En relación a los mecanismos de acceso al canal, el estándar define un periodo con contienda (SCP) y un periodo libre de contienda (CFP), aunque actualmente sólo el SCP está implementado por parte de los fabricantes.

CFP: consiste en un acceso al canal necesario para solicitar la asignación de dispositivos desde el Base node. Dependiendo del estado de uso del canal, el Base node puede conceder acceso al dispositivo solicitante para una duración específica o denegar la solicitud.

SCP: se trata de un acceso al canal que no requiere ningún tipo de arbitraje. Sin embargo, los dispositivos de transmisión necesitan respetar los límites de tiempo de SCP en una trama MAC. La composición de una trama MAC en términos de SCP y CFP se comunican en cada estructura como parte de la baliza.

A pesar de que actualmente solo se implementa el SCP, se está investigando sobre los beneficios potenciales de usar CFP para nuevos servicios de Smart Grids [14 AMS]. En el SCP se utiliza CSMA/CA como técnica de acceso al medio.

Anteriormente se ha comentado que en las redes PRIME pueden existir Service node denominados Switches, esto es debido a que en una subred, el Base node no puede comunicarse directamente con cualquier nodo, por lo que es necesario el uso de Switches node, los cuales se encargan de reenviar selectivamente el tráfico que se origina a partir de el o que está destinado a uno de los Service node dentro de su jerarquía de control de modo exista una ruta para poder hacer efectiva cualquier comunicación entre el Base node y los Service node de la red.

Para poder realizar con éxito dicha comunicación, cada Switch node mantiene una tabla con otros Switch nodes que estén conectados a través de él. El mantenimiento de esta información es suficiente para la conmutación, dado que el tráfico originado o finalizado en los Service node contiene el identificador de sus respectivos Switch node. Por tanto, la función de conmutación se simplifica ya que no es necesario mantener una lista con todos los nodos que conforman la red.

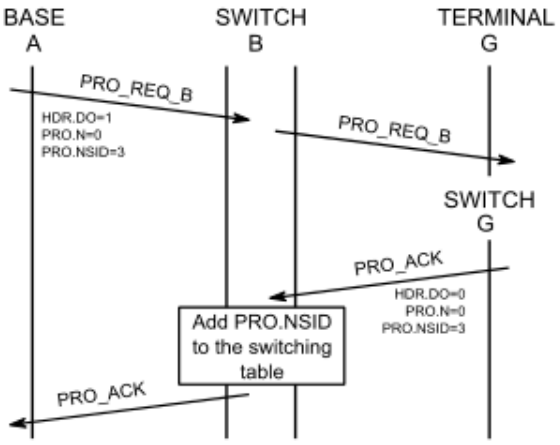



*Figura 3: Promoción a Switch*

En la figura 3 podemos ver que cuando el nodo G recibe el paquete PRO_REQ_B para su promoción, este cambia al modo Switch. El identificador de dicho Switch deberá ser el que contiene el campo (PRO-NSID,0) = (3,0) que es enviado por el Base node. La recepción y aceptación del PRO_REQ_B es reconocida mediante el PRO_ACK enviado por el nodo G. El Switch node intermediario (B) obtendrá el HDR.DO = 0, PKT.CTYPE = 3, PKT.SID=1 y el PRO.N=0 de forma que identifica que el PRO_ACK pertenece a uno de los Service node de su propia jerarquía de control. El nodo B deberá enviar el paquete PRO_ACK hacia el nodo base y este agregará el PRO.NSID a la Switching table.

Al comienzo se ha comentado que los Service node están inicialmente desconectados, para realizar la puesta en marcha inicial de un Service node, es necesario llevar a cabo el proceso de registro del mismo. Un Service node desconectado transmitirá un paquete de control (REG) para conseguir que el Base node le incluye en la subred. En esta etapa no se asigna ningún LNID o SID al Service node, pero sí se utiliza el SID del Switch node a través del cual se está solicitando la agregación a la subred.

En todas las solicitudes de registro aceptadas el Base node debe adjudicar un LNID (Local Node ID) único dentro del dominio del Switch node a través del cual se realiza el registro, es importante remarcar que los LNID tiene ámbito local, por lo que pueden existir varios Service node con el mismo LNID siempre y cuando cuelguen de distintos Switch node. La combinación del LNID (Local Node ID) con el SID (Switch ID) del nodo a través del cual se registró formaría el NID (Node ID) correspondiente para el Service node registrado.

El registro se realiza en un proceso de 3 pasos:

1. REG_REQ (Node -> Base)
2. REG_RSP (Base -> Node)
3. REG_ACK (Node -> Base)

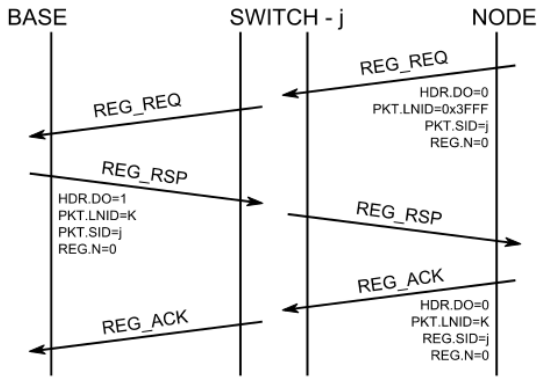



*Figura 4: Ejemplo de proceso de registro aceptado*

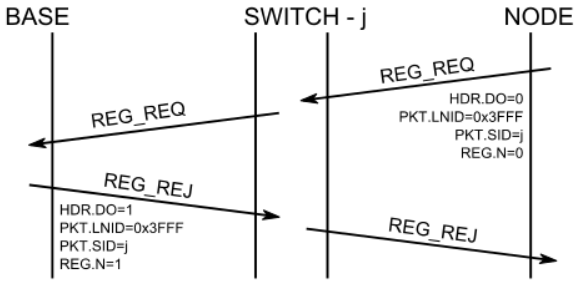



*Figura 5: Ejemplo de proceso de registro rechazado*

Según el estándar, cuando un Service node no puede alcanzar a cualquier otro Switch node, dicho nodo puede enviar tramas de promoción (promotion-needed) de modo que un terminal puede ser promovido y comenzar a funcionar como Switch. Durante este proceso, el nodo envía PNPDUs de modo que un terminal cercano puede ser promovido, comenzar a actuar como Switch y así el nodo mencionado puede solicitar comunicación hacia el Base node. Durante este periodo de tiempo, el Base node examina los promotion request para decidir si se acepta o no la promoción, el concentrador es el encargado de decidir que nodo se promoverá y en su caso, enviar una respuesta de promoción. El resto de nodos no recibirá ningún tipo de respuesta a la solicitud de promoción para evitar saturar la subred.

Es importante recalcar que cuando se reutilizan LSID que han sido liberados tras un proceso de degradación, el Base node no debe asignar el LSID hasta después de un tiempo establecido para asegurarse que todos los paquetes de retransmisión que usan ese LSID están fuera de la subred. De la misma manera, el Base node no debe reutilizar un LNID liberado en un proceso de keep alive hasta que haya pasado un determinado tiempo.

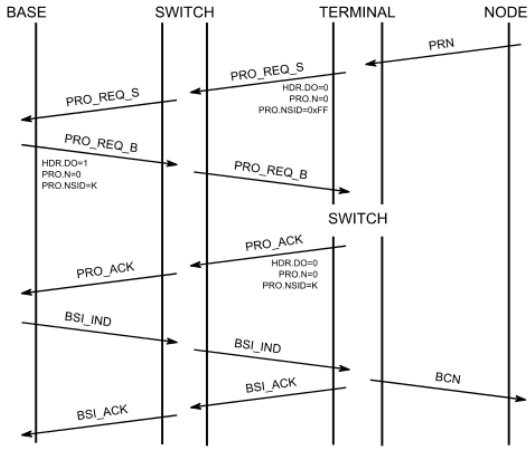



*Figura 6: Ejemplo de proceso de promoción iniciado por un Service Node (1)*

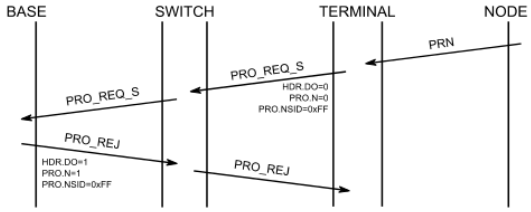



*Figura 7: Ejemplo de proceso de promoción iniciado por un Base Node (1)*

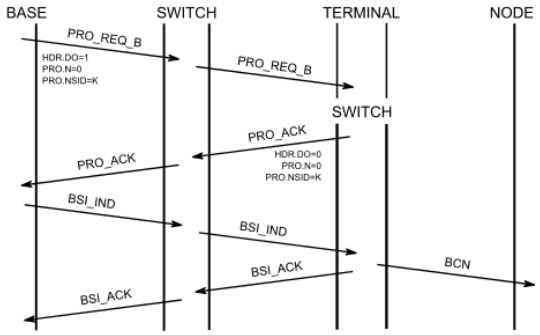



*Figura 8: Ejemplo de proceso de promoción iniciado por un Base node (2)*

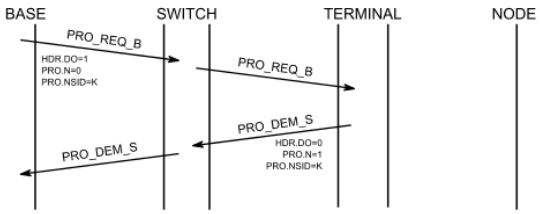



*Figura 9: Ejemplo de proceso de promoción iniciado por un Service node (2)*

En el proceso de degradación, el Base node o el Switch node pueden decidir el paso a terminal de un switch. Tras la finalización correcta de un proceso de degradación, el Switch node deja inmediatamente de enviar balizas y cambia su estado a terminal. El Base node podrá reasignar el LSID tras un tiempo estipulado por el estándar a otros terminales que soliciten la promoción.

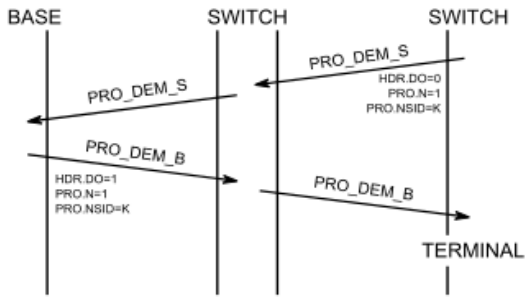



*Figura 10: Proceso de degradación iniciado por el Service node*

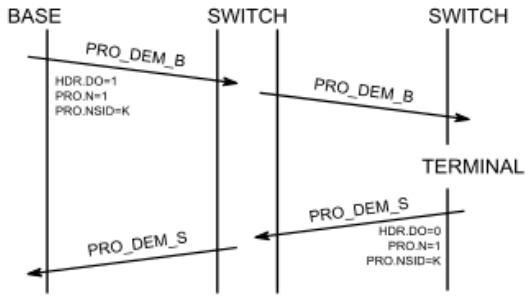



*Figura 11: Proceso de degradación iniciado por el Base node*

En PRIME el establecimiento de la conexión trabaja extremo a extremo. Todos los mensajes de establecimiento de la conexión utilizan paquetes de control. Cada establecimiento de conexión correcto establecido en la subred es asignado a un LCID. El Base node debe asociar un LCID único para un determinado LNID. Cuando la comunicación ha finalizado, cualquiera de los extremos de la conexión puede decidir el cierre de la misma en cualquier momento.

La capa de Control de Enlace Lógico (LLC), perteneciente a la capa de Convergencia, es responsable de manejar las conexiones lógicas. Identifica cada transacción con un número de identificación y realiza los procesos de control de flujo. El control de flujo está implementado en PRIME mediante el establecimiento de una Unidad Máxima de Transmisión (MTU) y usando un procedimiento de ventana deslizante. La MTU define la longitud en bytes del mayor paquete de datos que puede encapsular el nivel MAC. En caso de que la aplicación intente enviar un mensaje mayor, la capa LLC fragmenta el mensaje en varios paquetes, ninguno de los cuales será mayor que la MTU. Cada uno de estos paquetes está etiquetado con un identificador para que la parte receptora pueda re ensamblarlos.

Con respecto al procedimiento de ventana deslizante, PRIME establece diferentes valores permitidos para el Tamaño de Ventana (WS). El valor del WS puede jugar un rol importante en el rendimiento de la red en términos de latencia. Además, los dispositivos PRIME pueden implementar o no capacidades de Repetición de Solicitudes (ARQ) para asegurar la correcta recepción de todos los mensajes. Dado que los parámetros ARQ son negociados en la fase de conexión, este proceso funciona extremo a extremo entre el transmisor y el receptor final, siendo por tanto los switches transparentes al mismo.

En la capa de aplicación, DLMS/COSEM es el estándar utilizado sobre todas las tecnologías NB-PLC disponibles en el mercado. En concreto, COSEM (IEC 62056-61/62) es un perfil del protocolo de aplicación DLMS (IEC 62056-53) especialmente diseñado para la medición de energía. Como tal, DLMS/COSEM incluye modelos de datos para representar parámetros comunes relacionados con la energía junto con un protocolo de comunicación diseñado para transmitir este tipo de información.

## 2.2. Analizador de trazas (PrimeAnalytics)

PrimeAnalytics se trata de una herramienta de análisis PRIME desarrollada en el marco del proyecto OSIRIS (Optimización de la Supervisión Inteligente de la Red de dIStribución) por el equipo de investigación de la UC3M del cual formo parte. Permite extraer información de este tipo de redes de comunicación mediante el análisis de ficheros accesibles de manera telemática a través del concentrado.

Está desarrollado en Python bajo el Web framework Django que proporciona una gran fiabilidad y flexibilidad.

El funcionamiento del analizador se basa en tres ficheros que recibe como entrada proporcionado típicamente por las distribuidoras eléctricas, las cuales tiene acceso a los concentradores, los ficheros utilizados son:

- Fichero .csv con trazas de información
- Fichero .csv con eventos de topología
- Fichero .xml con el informe S11

Esta herramienta permite analizar el informe S11, analizar trazas de diferentes equipos PRIME (Circutor, Sagemcom, ZIV) y analizar los eventos de topología que surgen en la red.

La información se muestra en forma de gráficas, listados y buscadores, con los que se puede saber:

- Porcentaje de paquetes por tipo. Utilizando diferentes gráficas sobre el total de paquetes, sobre los paquetes que recibe el concentrador (uplink) y sobre los paquetes que envía el concentrador (downlink).
- Porcentaje de paquetes enviados/recibidos por el concentrador (uplink vs downlink)
- Evolución de la SNR a lo largo del tiempo para una determinada MAC
- Instantes en los que se produce el mayor número de desregistros en intervalos de 15 min.
- Dada una MAC devuelve el porcentaje de conexiones realizadas a cada uno de los contadores que funcionan como switch o repetidores.
- SNR medio para cada contador
- Número de peticiones de promoción a un switch provocado por cada uno de los contadores sin conexión directa al concentrador.
- Número de hijos de cada switch.
- Lista de nodos ordenados por número de desregistros de sus nodos hijo.
- Lista de nodos ordenados por su número de desregistros.

## 2.3. Lenguajes y frameworks de desarrollo

### 2.3.1. Python

Python es un lenguaje de programación multiparadigma, fácil de aprender, potente y con una semántica dinámica. Su enfoque es simple pero efectivo, del mismo modo que sus estructuras de datos son eficientes y de un alto nivel. Está basado en el paradigma de programación orientada a objetos. Se trata además de un lenguaje interpretado [16]

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1. [17].

Python usa un tipado dinámico y conteo de referencias para la administración de memoria. Se trata de un lenguaje diseñado con el objetivo de ser leído con facilidad. Como diferencia con otros lenguajes actuales, en Python se utilizan palabras donde otros lenguajes utilizan símbolos, facilitando así la legibilidad.

Python representa una de las opciones actualmente más utilizadas a la hora de desarrollar soluciones tanto de frontend como de backend, en gran parte debido a la gran comunidad Python que existe en la actualidad, debido a que se trata de un software libre. En su sitio Web podemos encontrar el intérprete Python, para las principales plataformas además de módulos, extensiones, programas y herramientas desarrollados por terceros que facilitan la mayoría de las implementaciones.

En la actualidad se mantienen dos ramas de Python con ciertas incompatibilidades entre sí, Python 2.7 y Python 3.x. El problema reside en que la versión 3.x no incluye la retro compatibilidad para algunos elementos de versiones anteriores, por lo que se recomienda el uso o actualización a la versión 3.x para evitar problemas de compatibilidad. [18]

El intérprete oficial de Python, CPython, que podemos encontrar en el sitio Web, está escrito en C, pero existen otras implementaciones alternativas [19]:

- Jython es la implementación de Python en la plataforma Java SE. Permite la integración total con aplicaciones Java existentes y el acceso a las clases y a la API de Java.

- IronPython es la implementación de Python en la plataforma .NET, permitiendo el acceso a las bibliotecas de esta plataforma y exponiendo el código de Python a otros lenguajes de .NET

- PyPy es un intérprete de Python escrito en Python. Su objetivo principal es mejorar el rendimiento de la ejecución manteniendo la máxima compatibilidad con CPython.

#### 2.3.1.1. Extensiones Python

- ❖ Psycopg2

El acceso a bases de datos utilizando el lenguaje de programación Python está estandarizado por la especificación Data Base API (DB-API), la cual se encuentra actualmente en la versión 2.0. (PEP 249: Python Databases API Specification v2.0). [20]

Para el manejo de la base de datos PostgreSQL se ha utilizado la extensión Psycopg2, se trata de un adaptador de PostgreSQL para el lenguaje de programación Python distribuida bajo los términos de la GNU Lesser General Public License.

Psycopg2 tiene una serie de características como pueden ser:

- Compatibilidad con versiones Python de 2.5 a 3.5
- Compatibilidad con versiones de PostgreSQL desde 7.4 a 9.5
- Compatibilidad con los procesos, los hilos pueden utilizar diferentes conexiones o compartir la misma conexión.
- Permite el envío y la recepción de forma asíncrona.

El uso de este módulo es sencillo, tan solo hay que seguir los siguientes pasos:

- Importar el conector.

- Conectarse a la base de datos.
- Abrir un Cursor.
- Ejecutar la consulta.
- Obtención de los datos.
- Cerrar el Cursor

❖ Statistics

Se trata de un módulo que proporciona las funcionalidades para realizar el cálculo estadístico de número. Dentro de él podemos encontrar funciones para calcular [21]:

- Promedios y medias de tendencia central.
- Medias de difusión

❖ Numpy

Numpy [22, 23] es una extensión de Python que ofrece un gran soporte para vectores y matrices, incluyendo una extensa biblioteca de funciones matemáticas de alto nivel para realizar diversas operaciones.

Se trata de un proyecto comunitario [24] plataforma con su última versión estable (1.10.2) lanzada el 14 de diciembre de 2015.

Aparte de operaciones matemáticas con vectores o arrays, Numpy ofrece la posibilidad de convertir estos datos a ficheros .csv con el fin de utilizarlos como entrada para otros programas.

❖ Subprocess

Módulo que permite activar nuevos procesos en la consola o terminal de nuestro sistema operativo. [25]

❖ 2.3.1.1.5. Http.client

Http.client es una extensión de Python que define las clases que implementan el lado del cliente en protocolos HTTP y HTTPS. [26]

### 2.3.2. Django

Se trata de un framework de código abierto para Python mantenido por la Django Software Foundation [27], respecta el patrón de diseño conocido como modelo MVC

(Modelo-Vista-Controlador). Está diseñado principalmente para facilitar una implementación y un despliegue rápidos de aplicaciones, facilitando la interacción del sistema implementado con la base de datos. Es fácilmente escalable y ofrece por defecto características de seguridad como la protección contra inyecciones SQL o un sistema de gestión de cuentas de usuario [28]

Este framework induce a un diseño modular de aplicaciones para fomentar la reutilización del código. Además, incluye un sistema de gestión (django-admin o manage.py) que permite, por ejemplo:

- Crear aplicaciones y módulos.
- Usuarios de administración.
- Gestión de la internacionalización del sistema.
- Lanzar un servidor Web de desarrollo.
- Gestionar la base de datos de forma simple y eficiente.

Incluye también otros componentes útiles como:

- Serialización de formularios.
- Validación de formularios según los tipos de datos admitidos y la obligatoriedad de los campos.
- Sistema de plantillas con herencia entre las mismas.

Aunque Django sigue un patrón de diseño MVC, según los desarrolladores del framework, esta terminología es debatible y prefieren referirse a un patrón de diseño Modelo-Plantilla-Vista (MTV o Model-Template-View), correspondiéndose los componentes vista y controlador del patrón MVC a los componentes plantilla y vista respectivamente (el componente modelo no varía de nombre). Para facilitar la comprensión de los lectores no familiarizados con Django, en esta memoria se hace referencia a la terminología estándar de diseño MVC.

### 2.3.3. PostgreSQL

PostgreSQL es un potente sistema de base de datos objeto-relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de fiabilidad e integridad de datos. Se ejecuta en los principales sistemas operativos utilizados en la actualidad (Linux, UNIX y Windows).

PostgreSQL utiliza un modelo cliente/servidor con la diferencia que usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. De esta forma un fallo en uno de los procesos no afectará al resto y el sistema continuará funcionando.

El desarrollo de PostgreSQL, al igual que muchos proyectos de código abierto, es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada. Dicha comunidad es denominada PGDG (PostgreSQL Global Development Group) [29]

Es totalmente compatible con ACID, al igual que cumple con el estándar ANSI-SQL:2008 [30]. Incluye una biblioteca de funciones estándar con cientos de funciones integradas que van desde las operaciones matemáticas básicas, operaciones con string para criptografía y compatibilidad con Oracle. [30]

## 2.4. Herramientas para el despliegue

### 2.4.1. Docker

Docker [31] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux.

Implementa una API de alto nivel para proporcionar contenedores livianos que ejecutan procesos de manera aislada.

Construido sobre el kernel de Linux (principalmente cgroups y namespaces), un contenedor Docker, a diferencia de una máquina virtual, no requiere incluir un sistema operativo independiente. En su lugar, se basa en las funcionalidades del kernel y utiliza el aislamiento de recursos (CPU, memoria, red…) y namespaces separados para aislar de vista la aplicación del sistema operativo.

Utilizando los contenedores de Docker podemos simplificar la creación de sistemas altamente distribuidos, permitiendo que diferentes aplicaciones, la tarea de los trabajadores (workers) y otros procesos funcionar de manera autónoma en una única máquina física o en varias máquinas virtuales.

Además, el peso de un sistema basado en Docker no tiene comparación con cualquier sistema de virtualización más convencional. Por ejemplo, una imagen de Ubuntu corriendo en VirtualBox puede llegar a pesar 1Gb, por lo que un contenedor que contenga Ubuntu con Apache y una aplicación web puede llegar a los 180Mb, lo que demuestra un significativo ahorro a la hora de almacenar diversos contenedores que podemos desplegar posteriormente.

Dentro de Docker podemos utilizar archivos y herramientas como:

- Dockerfile, se trata de un archivo que reconoce Docker y que contiene una serie de instrucciones para automatizar el proceso de creación de un contenedor. En este

archivo se incluye todo lo necesario en nuestro contenedor para que cada vez que ejecutemos el script de construcción obtengamos un contenedor completamente nuevo y actualizado.

- Docker Engine [31], se trata de una aplicación cliente-servidor formada por un servidor que se encarga de llamar al daemon, una API que especifica las interfaces de programas que se pueden utilizar para dar instrucciones al daemon y una interfaz de línea de comandos (CLI) para el cliente.

- Docker Compose [32], es una herramienta para la definición y ejecución de aplicaciones Docker multi-contenedor. Para ello, se utiliza un archivo de composición para configurar los servicios de aplicación. Después, con un solo comando, se crean e inician todos los servicios en función de la configuración definida en el Compose

- Docker Machine [33], es una herramienta que permite crear un sistema con la misión de albergar contenedores de Docker aunque no utilicemos Linux. Puede desplegarse en VirtualBox, Amazon EC2 o Digital Ocean entre otras.

En definitiva, la utilización de Docker reduce el tiempo empleado en configurar nuevos entornos o en solucionar problemas asociados con el uso de entornos diferentes, permite la transferencia de aplicaciones desde entornos de desarrollo a entornos de producción, además de que las aplicaciones basadas en contenedores facilitan la implementación, la identificación de problemas y el retorno a fases anteriores para remediar fallos.

## 2.5. Herramientas para la paralelización de bases de datos

### 2.5.1. PgPool

Es un middleware que se sitúa entre los servidores de PostgreSQL y un cliente de base de datos PostgreSQL, PgPool ofrece características como [34]:

- Agrupación de conexiones, se encarga de mantener las conexiones establecidas a los servidores PostgreSQL y los reutiliza cada vez que se crea una nueva conexión con las mismas propiedades (mismo usuario, base de datos y protocolo) mejorando el rendimiento global del sistema.

- Replicación, esta función permite la creación en tiempo real de una copia de seguridad en 2 o más grupos PostgreSQL, de forma que el servicio pueda continuar sin interrupción si uno de los grupos falla.

- Balanceo de Carga, partiendo de una replicación de base de datos en la que se puede ejecutar en modo replicación o en modo esclavo/maestro, a la hora de realizar una SELECT en cualquier servidor, se obtendrá el mismo resultado.

PgPool-II aprovecha la función de replicación para reducir la carga en cada servidor de PostgreSQL, repartiendo las consultas SELECT entre los servidores disponibles obteniendo un mejor rendimiento del sistema.

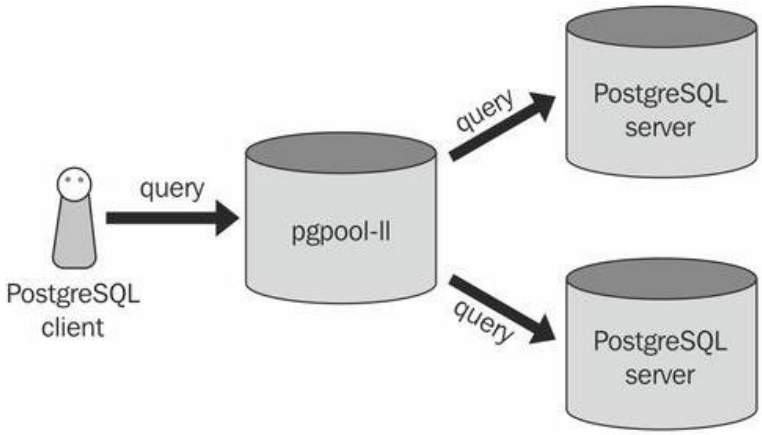



*Figura 12: Funcionamiento PgPool-II*

PgPool-II habla los protocolos de frontend y backend de PostgreSQL, y pasa las conexiones entre ellos. De ese modo, una aplicación de base de datos (frontend) cree que PgPool-II es el verdadero servidor de PostgreSQL, y el servidor (backend) ve a PgPool-II como uno de sus clientes. Debido a que PgPool-II es transparente tanto para el servidor como para el cliente, una aplicación de base de datos existente puede empezar a usarse con el sin realizar prácticamente ningún cambio en su código fuente.

En un escenario ideal, el rendimiento de lectura podría mejorar proporcionalmente al número de servidores PostgreSQL.

PgPool-II permite limitar el exceso de conexiones, en PostgreSQL hay un límite máximo en el número de conexiones simultáneas, el cual una vez alcanzado rechaza las nuevas conexiones. Se puede aumentar el número de conexiones pero aumenta el consumo de recursos afectando negativamente al rendimiento del sistema. La ventaja al usar PgPool-II es que una vez alcanzado el número máximo, las nuevas conexiones en vez de rechazarse se pondrán en lista de espera.

La utilización de consultas en paralelo permite la división los datos en varios servidores, por lo que una consulta puede ejecutarse en todos los servidores al mismo tiempo reduciendo el tiempo de ejecución total. Este tipo de consulta es la más apropiada para la búsqueda de datos a gran escala.

PgPool-II transmite mensajes entre el backend y el frontend. La aplicación de base de datos (frontend) ver a PgPool-II como el servidor PostgreSQL actual, y el servidor (backend) ve a PgPool-II como a uno de sus clientes, por lo que PgPool es transparente en la conexión servidor-cliente. A continuación se muestra la arquitectura interna de PgPool y una visión sobre la integración en un sistema de base de datos.

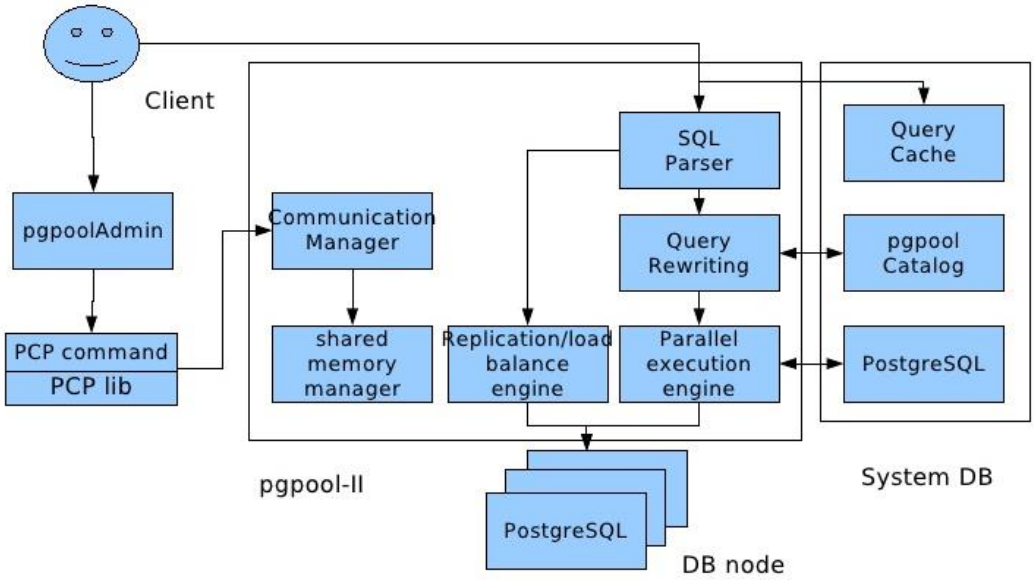



*Figura 13: Arquitectura PgPool-II*

### 2.5.2. PgShard

Se trata de una extensión sharding para el sistema de bases de datos PostgreSQL, permite la fragmentación y replicación de las tablas con el fin de obtener una alta disponibilidad y el escalado horizontal.

Dicha extensión funciona con sentencias SQL por lo que no requiere ningún cambio adicional en el código.

Dado que se trata de una extensión independiente, permite abordar muchos casos de uso basados en NoSQL. Además, permite el análisis en tiempo real de los datos.

El funcionamiento básico de PgShard es tomar una o más tablas de la base de datos y distribuirla en fragmentos alojados en varios servidores. Las filas de estas tablas se distribuyen sobre el fragmento utilizando una función hash basada en la clave principal (Primary Key) definida en la tabla.

Para orquestar el correcto funcionamiento se hace uso de un nodo Maestro el cual contiene información sobre los fragmentos distribuidos, ya que es este nodo el que se

encarga, de manera transparente, de que las inserciones, actualizaciones y consultas se realizan sobre los fragmentos correctos.

Una característica a destacar es la posibilidad de crear varias réplicas de uno o varios fragmentos, de esta forma se aumenta la redundancia de la base de datos, lo cual se traduce en un aumento de la disponibilidad si cada réplica se coloca en servidores diferentes.

Las consultas o Selects sobre la base de datos se distribuyen en función del condicional "Where" utilizando la clave hash como condicionante, de manera que el nodo maestro solo realiza la consulta contra los fragmentos que tienen datos compatibles con la consulta que se lanza.

El uso de consultas en paralelo mediante la función hash permite la reducción de tiempo comparado con las consultas tradicionales, puesto que en la consulta en paralelo solo se ha de comprobar un porcentaje de la tabla, algo importante cuando el número de filas en la tabla es muy elevado.

El funcionamiento de las consultas en paralelo es totalmente transparente para el usuario, simplemente hay que lanzar la consulta hacia el nodo maestro y él se encarga de realizar la paralelización de la consulta.

Hay que destacar dos detalles importantes de PgShard, la primera es que como las filas se distribuyen en los fragmentos utilizando una función hash, solo se admiten consultas en las que se utilicen comparaciones de igualdad con la clave primaria. En el caso de búsquedas por rango, se comprueban los datos de todos los fragmentos. La segunda es que solo se admite un nodo maestro, por lo que no es apto para aplicaciones que necesitan un gran número de conexiones [35] [36].

### 2.5.3. Citus

Se trata de una extensión del sistema de base de datos PostgreSQL que surge como actualización de PgShard [37] [38]. Con el se puede realizar un escalado horizontal a través de múltiples máquinas o nodos denominados "Workers", ofreciendo replicación y fragmentación de tablas.

Citus utiliza un motor de búsqueda paralelizando las consultas SQL entrantes al nodo denominado "Master", a través de los Workers para permitir respuesta en tiempo real a grandes conjuntos de datos.

Una de las ventajas de esta extensión es que permite seguir utilizando las características y herramientas existentes de PostgreSQL.

Utiliza una arquitectura modular que se basa en la distribución de los datos a través de un clúster de Workers, las consultas SQL entran al nodo Master y este las procesa en paralelo a través de los Workers. En dicha arquitectura el Master, en lugar de almacenar los datos, almacena tablas de metadatos, las cuales permiten realizar un seguimiento de todos los Workers del clúster así como las ubicaciones de los fragmentos en cada uno de ellos.

Cada fragmento se replica en al menos dos Workers, aunque este número se puede modificar, de forma que la pérdida de alguno de los Workers no tiene impacto sobre la disponibilidad de los datos. Esta arquitectura permite además la agregación de nuevos Workers en cualquier momento para aumentar la capacidad y potencia de procesamiento de la agrupación.

Citus permite dos métodos de distribución de los datos a través de los fragmentos, distribución “hash” y distribución “append”

El método de distribución hash permite realizar búsquedas rápidas basándose mediante clave-valor. Para utilizar este método, se ha de elegir la columna de la tabla que se utilizará como columna de distribución.

El método de distribución append permite la distribución de datos de forma ordenada a través de los fragmentos creados en los Workers. Al igual que en la distribución hash, se ha de elegir la columna de la tabla que se utilizará como columna de distribución.

Los pasos de configuración de ambos métodos de distribución se describirán en los capítulos siguientes.

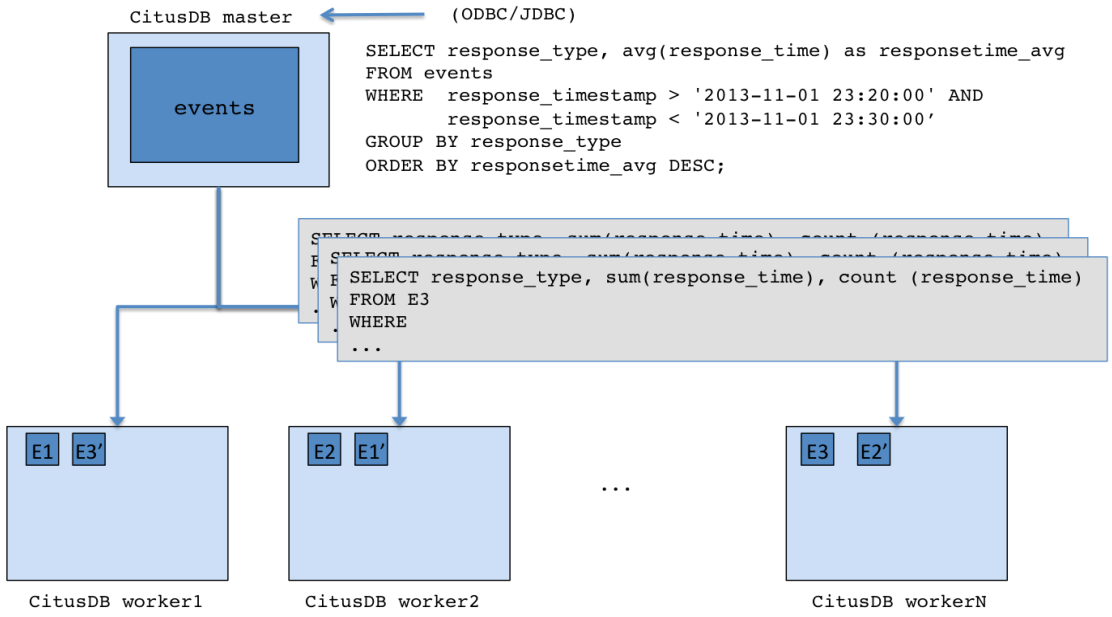



*Figura 14: Arquitectura modular Citus*

El procesamiento de consultas en esta extensión implica dos componentes importantes:

- Planificador de consultas distribuidas: es el encargado de recibir las consultas "SELECT" de PostgreSQL y convertirlas en consultas en paralelo sobre los Workers. Una vez paralelizadas las consultas, divide la consulta en dos, una es la consulta realizada al Master y otra las consultas que se lanzan a los fragmentos de los Workers. Se encarga de enrutar las consultas a los Workers correspondientes de forma que los recursos se utilizan de manera eficiente.

- Ejecutor de consultas distribuidas: los ejecutores se conectan a los Workers para enviar las tareas que se les han asignado y supervisar la ejecución. Si una tarea no pudiera ser asignada a un trabajador, el ejecutor, de forma dinámica re asigna la tarea a las réplicas de otros trabajadores. Existen dos tipos de ejecutores, en tiempo real y seguimiento de tareas.

- Planificador y Ejecutor PostgreSQL: envía las consultas a los Workers, los cuales las procesan como consultas PostgreSQL normales, y se encarga de recibir la respuesta de cada uno de ellos para devolverla al Master.

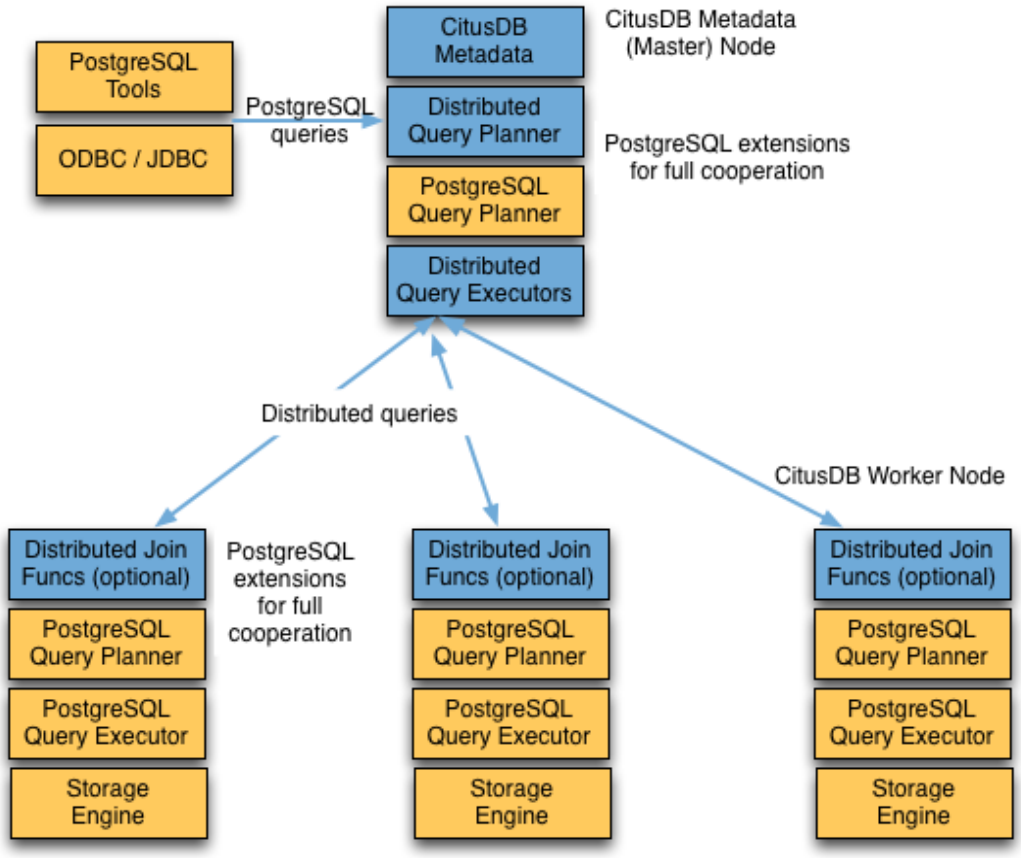



*Figura 15: Arquitectura Interna Citus*

En los siguientes capítulos se abordará en profundidad la configuración y funcionamiento de la extensión.

## 2.6. Plataformas Cloud

### 2.6.1. FIWARE

Se trata de un proyecto Europeo dentro de la estrategia Horizonte 2020, financiado de forma conjunta entre empresas privadas y fondos públicos, cuyo objetivo es promover la innovación en el sector IT y desarrollar los estándares de las nuevas tecnologías como el Cloud Hosting, Big Data, IoT, etc [39].

FIWARE busca ofrecer una plataforma para desarrolladores con interfaces de programación que permitan a dichos desarrolladores construir aplicaciones capaces de resolver problemas reales y concretos de los consumidores.

FIWARE es una plataforma OpenSource que permite la integración de aplicaciones y soluciones desarrolladas por terceros a través de un proceso sencillo, seguro y de bajo coste. Facilitado gracias al desarrollo de API´s con tecnologías OpenSource.

Todas las herramientas facilitadas por FIWARE están listas para ser utilizadas en FIWARE Lab, del que se hablará más adelante.

Como hemos comentado al principio, FIWARE nace en el seno del Horizonte 2020, y se marca como sectores preferentes las Smart Cities, E-Health, E-Learning, Energía y medio Ambiente, Transporte, Agricultura, Media y Contenido y Logística, todos ellos enfocados a la mejora competitiva y a la calidad de vida en Europa.

El objetivo de esta plataforma es recoger el testigo de la digitalización y utilizarlo para impulsar la creación de un ecosistema sostenible basado en la integración de las tecnologías de internet, basándose para ello en el desarrollo de una tecnología propia gratuita en la que encontraremos un set de interfaces de programación de aplicaciones (APIs)

Actualmente hay dos elementos a tener en cuenta para entender el propósito de FIWARE.

Por un lado tenemos el desarrollo tecnológico, que está contribuyendo a la creación de un nuevo contexto donde las aplicaciones a través de la nube y el Internet de las Cosas, son capaces de facilitarnos datos en tiempo real.

Por otro lado, viendo la evolución de la tecnología y el mundo de las comunicaciones, todo apunta a que en un futuro no muy lejano Internet va a estar estrechamente relacionado con el compartimiento de datos. Por lo que es necesario un entorno colaborativo donde las aplicaciones sean desarrolladas por terceros e integradas en plataformas compartidas, con el

fin de obtener mayor eficiencia, competitividad y un mejor servicio tanto a ciudadanos como a consumidores.

FIWARE se sustenta sobre los siguientes pilares:

- FIWARE API, la plataforma FIWARE proporciona un conjunto de APIs (Application Programming Interfaces) simple pero potente, cuyo objetivo es facilitar el desarrollo de aplicaciones inteligentes en diversos sectores. Las especificaciones de estas APIs son públicas y libres de royalties, aparte hay una implementación de referencia pública en código abierto para cada uno de los componentes, de forma que se facilita que los proveedores de FIWARE puedan surgir rápidamente en el mercado con propuestas de bajo coste.

- FIWARE Lab, se trata de un entorno no comercial que pretende ser un punto de encuentro para la innovación y la experimentación basadas en tecnologías FIWARE. El objetivo del Lab es que tanto empresas como particulares puedan probar tanto la tecnología como las aplicaciones en este entorno, explotando datos abiertos publicados por terceros. La topología de Fiware Lab se compone de una red de nodos distribuida geográficamente aprovechando una gran gama e infraestructuras experimentales.

- FIWARE Accelerate, consiste en un programa de aceleración que tiene como objetivo promover la asimilación y uso de las tecnologías basadas en FIWARE por parte de los integradores de soluciones, los desarrolladores de aplicaciones y especialmente centrado en PYMEs y empresas de nueva creación. En 2014 la UE puso en marcha una campaña financiada con más de 80 millones de € para el apoyo de PYMES y empresas que desarrollen aplicaciones basadas en FIWARE.

- FIWARE Mundus, el programa Mundus está diseñado para llevar la experiencia FIWARE a los principales actores del sector TIC y obtener una coordinación con los gobiernos en diferentes partes del mundo aprovechando que aunque el proyecto FIWARE nació en Europa, ha sido diseñado con una ambición global.

- FIWARE iHubs, la red iHub tiene por objeto apoyar la creación y el funcionamiento de nodos iHubs en todo el mundo, demostrando la actitud FIWARE de pensar globalmente pero actuar localmente.

FIWARE pone a nuestra disposición una serie de recursos en forma de máquinas virtuales, denominadas Flavours. Estas máquinas tienen diferentes cuotas de memoria RAM, disco duro y número de CPUs, en función de las necesidades que requiera la aplicación a desarrollar. [39]

### 2.6.2. Microsoft Azure

Microsoft Azure, (antes conocido como Windows Azure o Azure Services Platform) es una plataforma ofrecida como servicio o SaaS (Software as a Service) [38] alojada en los Data Center de Microsoft. Anunciada en el Professional Developers Conference de Microsoft (PDC) del 2008 en su versión beta, fue el 1 de Enero de 2010 cuando se ofreció como versión comercial [48]

Azure es compatible con la mayor selección de sistemas operativos, lenguajes de programación, marcos, herramientas, bases de datos y dispositivos. Permite la la ejecución de contenedores Linux gracias a la integración con Docker.

Principalmente Azure ofrece 3 productos de gran importancia en el actual mundo de las TICs:

- IaaS (Infraestructure as a Service) [40], se trata de una serie de servicios orientados a que el usuario tenga el control total de la infraestructura virtual. En este apartado se incluye todo lo relacionado con servidores (máquinas virtuales) donde se puede escoger entre una gran variedad de sistemas operativos, el número de núcleos de procesamiento, tamaño de la memoria RAM y la capacidad de almacenaje en discos virtuales. Azure cuenta con una larga lista de máquinas virtuales con diferentes características para poder cubrir cualquier tipo de necesidades.

- PaaS (Platform as a Service) [41], con Azure nos encontramos una plataforma ya creada y gestionada por nosotros, permitiendo escalar recursos en función de las necesidades de las aplicaciones subidas a Azure. Azure permite el uso de frameworks como Django, lo cual ofrece más opciones que si fuera una plataforma basada en .Net, ASP o SQL Server.

- SaaS (Software as a Service) [42], en este apartado nos encontramos con servicios en los cuales la infraestructura y la plataforma están debajo de una capa de abstracción. En ella, el cliente consume directamente las aplicaciones en formato de servicios. Un ejemplo de estos servicios pueden ser servicios de Big Data como Hadoop, servicios de comunicaciones como Biztalk o servicios de difusión como Service Bus.

### 2.6.3. IBM SoftLayer

SoftLayer pertenece a la división de Servicios Cloud de IBM, la cual al igual que Azure ofrece los productos de IaaS, PaaS, SaaS y BPaaS (Bussines Process as a Service).

Actualmente cuenta con 40 Data Centers repartidos por todo el mundo ofreciendo así una red privada global. La presencia en Europa se centra en ciudades como Amsterdam, Portsmouth, Ehningen, Winterthur, Montpellier, Lisbon, Barcelona, Londres, Paris y Frankfurt.

Este cloud permite el denominado HybridCloud que junta el Cloud Privado & IT tradicional con el Cloud Público consiguiendo así:

- Maximizar el retorno de la IT existente.
- Utilizar las cargas de trabajo que se adaptan mejor a determinadas infraestructuras o proyectos.
- Mitigar los riesgos mejorando la eficiencia.
- La absorción de picos de trabajo sin necesidad de incrementar la inversión de capital.
- Implementar nuevas funcionalidades con la mayor rapidez para responder a las expectativas del mercado.

Softlayer ofrece una API en la cual existen más de 2.200 scripts que dan acceso a más de 200 servicios disponibles, permite mejorar el control y la agilidad de la infraestructura, optimizar los recursos y el retorno de la inversión, aparte de soportar lenguajes como C#, PERL, PHP, Python, REST, Ruby, etc.

Aparte, incluye funcionalidades como:

- Despliegue automático de servidores.
- Reboots & reloads.
- Gestión de tickets.
- Configuración del hardware.
- Carga de software.
- Gestión de DNS & network.
- Gestión de almacenamiento.
- Gestión del monitoring.

El principal punto que diferencia a SoftLayer del resto de Clouds es su estructura formada por tres bloques, red pública, red de gestión y red privada [43] [44].

### 2.6.4. Google Cloud Platform

Google Cloud Platform [45] se trata de una plataforma que permite probar, crear e implementar diversos servicios y aplicaciones entre los cuales destacan:

- Compute Engine: se trata de un servicio de IaaS ofrecido por Google, en él se pueden gestionar diferentes máquinas virtuales, permitiendo a su vez el uso de contenedores Docker.

- App Engine: es el producto PaaS que ofrece esta plataforma, permite el desarrollo de funcionalidades encargando a la plataforma las funciones de escalabilidad y disponibilidad. Permite desarrollo en lenguajes como Python, Java, PHP o Go entre otros.

- Cloud Storage: GCP ofrece almacenamiento de datos con la ventaja de replicación automática en varios lugares del mundo para ofrecer redundancia de información y disponibilidad.

- Google Cloud Platform, es desde un punto de vista técnico, una plataforma en la que se han reunido todas las aplicaciones de desarrollo web ofrecidas por Google.

### 2.6.5. Amazon Web Services

Amazon Web Services (AWS) [46] se trata de una recopilación de servicios para la computación en la nube que forman, en su conjunto una plataforma ofrecida por Amazon.com y considerado como pionero en su segmento.

AWS se encuentra repartido en 11 regiones geográficas, cada región opera dentro de un solo país por lo que todos sus datos y servicios permanecen dentro de dicha región. Dentro de cada una hay disponible varias zonas de disponibilidad, donde se alojan los centros de datos. Estas zonas están aisladas de forma que se evita la propagación de errores y/o cortes al resto de zonas.

La unidad básica de AWS es la EC2 (Amazon Elastic Compute Cloud), permite a los usuarios el uso de recursos virtuales en los cuales ejecutar sus propias aplicaciones y ofreciendo el servicio de pago por utilización.

EC2 se basa en técnicas de virtualización, soportando diversos sistemas operativos. Posee una interfaz de servicios web para iniciar y configurar los servicios de forma sencilla, proporciona un control completo de los recursos y reduce el tiempo de inicio de los servidores.

Dentro de EC2 existe una amplia variedad de instancias según los recursos que se necesiten. Van desde instancias como la t2.nano (1 CPU, 0.5 GB RAM) hasta instancias x1.32xlargue (128 CPU, 1952 GB RAM). Para este tipo de instancias no existe cuota mínima, solamente se tarifica el tiempo en el que se estén usando los recursos.

Aparte de la EC2, AWS ofrece una serie de productos entre los cuales destacan:

Almacenamiento en la Nube:

a) Amazon Simple Storage Service; sistema de almacenamiento de objetos utilizando una API para poder acceder a ellos desde cualquier lugar.
b) Amazon Elastic File System: almacenamiento de datos para disponer de ellos desde una o varias instancias EC2
c) Amazon Elastic Block Storage: almacenamiento por bloques en volúmenes virtuales.

Bases de datos:

a) AWS Database Migration Service: servicio para facilitar la migración de bases de datos a AWS.
b) Amazon RDS: base de datos relacional administrada en la nube
c) Amazon Aurora: base de datos compatible con MySQL
d) Amazon DynamoDB: base de datos NoSQL

Redes:

a) Amazon VPC: servicio de nube virtual privada para aislar recursos en la nube con su propia red virtual privada.
b) Elastic Load Balancing: servicio de distribución automático del tráfico de aplicaciones sobre varias instancias Amazon EC2

Análisis de datos:

a) Amazon Elastic MapReduce: servicio Hadoop pensado para procesar grandes cantidades de datos.
b) Amazon Kinesis: servicio para datos de streaming
c) Amazon Machine Learning: producto para hacer uso de la tecnología del aprendizaje automático.

A continuación se muestran dos tablas comparativas de todas las infraestructuras Cloud expuestas.

| Nombre | Cuenta de prueba | Recursos Hardware | | | Recursos Software | | Coste Cuenta Premium | Notas |
|---|---|---|---|---|---|---|---|---|
| | | CPU | RAM | HD | Cloud (OpenStack) | Big Data | | |
| **Microsoft Azure** | Comunidad Dreamspark | Azure ofrece acceso a todos sus recursos durante la cuenta de prueba (170€ como máximo en total)<br>Link máqunias virtuales | | | Integración con OpenStack (link) | > Haddop<br>> HD Insight (Apache Spark)<br>> Analisis con Data Lake<br>> Procesamiento de datos SQL | Tarificación por minutos.<br>Link a precios | > SDK para .NET, Java, Node.js, PHP, Python, Ruby….<br>> Ej de interacción con servicios de Azure<br>> Formación y documentación<br>> Seminarios |
| **Amazon AWS** | 12 meses (750h/mes) | Instancia estándar t2.micro EC2 (Gratis) | | | Integración con OpenStack (link) | > Elastic MapReduce (EMR), infraestructura Hadoop alojada.<br>> Almacenamiento de datos (Redshift)<br>> Transmisión de datos en tiempo real (kinesis)<br>> Apache Spark en EC2 (link) | Tarificación por horas.<br>Link a precios<br><br>t2.micro(0.013$)<br>m3.medium(0.067$)<br>m3.largue(0.133$) | > Instancias con Linux, RHEL,SLES,Windows<br>> AWS Lambda (Python, Java y JavaScript)<br>> Tutoriales<br>> Cursos de formación<br>> Laboratorio<br>> Certificación AWS |
| | | X1 | 1 GB | 30GB EBS | | | | |
| | | Instancia estándar m3.medium EC2 (De pago) | | | | | | |
| | | X1 | 3.75 GB | 4 GB SSD | | | | |
| | | Instancia estándar m3 . largue EC2 (De pago) | | | | | | |
| | | X2 | 6.5 GB | 32 GB SSD | | | | |
| **FIWARE** | Necesita aprobación | X1 | 4GB | 20 GB | Basado en OpenStack (link) | > GE de Data/Context Management como CKAN,Kurento,Cosmos... | No hay cuenta premium, pero necesita aprobación | > Poca documentación y/o ejemplos para iniciarse. |
| **Google Cloud Platform** | 2 meses (200$ máximo) | Link del listado de máquinas disponibles | | | Kubernetes (link) (Container Engine) | > Apache Hadoop (aprox 367$/mes)<br>> Hadoop&Spark<br>> DataProc en fase beta | Tarificación por minutos. (link) | > Documentación disponible.<br>> S.O. como CentOS, Debian, RHEL,SUSE,Ubutnu y windows.<br>> Lenguajes, Node.js, Python, Ruby y Go. |
| **IBM SoftLayer** | 1 mes | X1 | 1GB | 25 GB | OpenStack 1<br>OpenStack 2<br>OpenStack 3 | > Apache Hadoop<br>> Apache Spark | Tarificación por horas o subscripción mensual. (link) | > Soporta C#, PERL, PHP, Python,REST,Ruby y VB.net) |

*Tabla 1: Comparativa arquitecturas Cloud*

| | Amazon AWS | Microsoft Azure | Google Cloud Platform | FIWARE | IBM SoftLayer |
|---|---|---|---|---|---|
| **Backups** | 3 copias por defecto | 3 copias por defecto | Backups en todas las plataformas | Sin información | Sin información |
| **Disponibilidad** | > 11 centros de datos<br>> 37 puntos de distribución | > 20 centros de datos<br>> 32 puntos de distribución | > 4 centros de datos<br>> 160 puntos de distribución | > 5 centros de datos<br>> 20 puntos de distirbución | > 26 centros de datos<br>> 22 puntos de distirbución |
| **Marketplace** | 2400 apps | 707 apps | 160 apps | 46 apps | 375 apps |
| **Servidores** | 53 tipos | 25 tipos | 18 tipos | 4 tipos | Sin información |
| **Tipos de HD** | SSD y mecánicos | SSD y mecánicos | SSD y mecánicos | SSD y mecánicos | SSD y mecánicos |
| **Servicios de base de datos en la nube** | MySQL, Oracle, PostgreSQL y SQL Server | SQL Server | MySQL + NoSQL | Sin información | MySQL, Oracle, PostgreSQL y SQL Server |
| **Seguridad (ISO27001)** | Si | Si | Si | Si | Si |
| **Migración servidores virtuales** | Vmware e Hyper-V | Hiper-V | Aun no lo soporta | Sin información | Citrix XenServer, Hyper-V y VMware |
| **Indisponibilidad** | 1,02 horas al año | 33,22 horas al año | 9,42 horas al año | Sin información | Sin información |
| **Apache Hadoop** | SI | Si | Si | Si | Si |

*Tabla 2: Comparativa sobre características de las Cloud*

Tras la realización de este estudio del arte se han llegado a una serie de conclusiones, las cuales se engloban en dos grupos.

En el primer grupo se llegó a las siguientes conclusiones:

- El estándar PLC utilizado debe ser PRIME, esto es debido a que el analizador de trazas está basado en dicho protocolo.

- Es necesario conocer y manejar correctamente las herramientas Docker, Django y PostgreSQL así como el lenguaje de programación Python debido a que el analizador de trazas se desarrolló utilizando todo ello.

En el segundo grupo se llegó a las siguientes conclusiones:

- Por un lado, en lo referente a la paralelización de bases de datos se eligió la extensión Citus debido a su disponibilidad en una imagen Docker y a la documentación de configuración ofrecida en su web oficial. No obstante en un primer momento se comenzó a implementar PgPool-II pero finalmente se vio que se había descontinuado la opción de paralelización, centrándose únicamente en el uso del clúster para aumentar la disponibilidad.

- Por otro lado en lo relacionado con la computación en la nube, tras analizar las diferentes Clouds expuestas se optó por el uso de Amazon Web Services, principalmente porque tratarse de una infraestructura madura, con amplia documentación sobre su uso y por ofrecer la cuenta de prueba con mayor duración.

# Capítulo 3. Diseño

El objetivo de este capítulo es describir el proceso de diseño de la solución de paralelización así como la integración para el correcto funcionamiento en el analizador de trazas basado en el Framework Django y el sistema de bases de datos PostgreSQL para su posterior subida a la plataforma Cloud. Para ello, se presentan en primer lugar las distintas herramientas necesarias para el desarrollo, los distintos requisitos funcionales así como las limitaciones y problemas surgidos a la hora de realizar la implementación. Para intentar ofrecer una imagen clara de las implementaciones que se pretenden conseguir se exponen una serie de figuras para ilustrar el funcionamiento actual de la herramienta y el funcionamiento esperado al aplicar las técnicas de optimización elegidas. En último lugar, se exponen las principales decisiones técnicas y de diseño que se han tomado de cara al desarrollo del trabajo, descrito en el capítulo siguiente.

## 3.1. Requisitos y restricciones

Los requisitos que definen este proyecto vienen inicialmente establecidos por los antecedentes que ya fueron presentados en el capítulo 1. El objetivo final es la evaluación de dos técnicas, paralelización de bases de datos y computación en la nube, para conseguir la optimización de la herramienta de análisis de trazas en redes PRIME.

Puesto que partimos de una herramienta funcional, tenemos una serie de requisitos imprescindibles para la correcta implementación del sistema:

**Requisito I: Utilización de Python, Docker y Django como herramientas de desarrollo.**

La herramienta que se pretende optimizar está desarrollada bajo el entorno de desarrollo Django que, como se vio en el capítulo 2, es un Web framework escrito en Python y que utiliza contenedores Docker para la instalación de la base de datos PostgreSQL. Dado que las funcionalidades del analizador están escritas en Python, la utilización de este lenguaje de programación para modificar los archivos de la herramienta, así como la creación de nuevos scripts para la correcta utilización de Citus se toma como requisito debido a que facilita la integración del código desarrollado y su posterior mantenimiento.

En lo referente a la utilización de Django, es imprescindible ya que de lo contrario habría que rehacer el analizador de trazas bajo otro tipo de framework. No obstante, la utilización de Django obligará a realizar una serie de scripts y modificaciones en el código inicial para solventar algunos problemas a la hora de integrar la extensión de Citus en la herramienta que se explicarán más adelante.

Por otro lado, la utilización de Docker también es esencial, en primer lugar porque la herramienta hace uso de imágenes Docker para la configuración de Django y PostgreSQL que evitan la instalación fija y facilita su portabilidad a entornos Cloud u otros equipos. En segundo lugar, debido a que Citus proporciona también una imagen Docker, lo cual facilita su integración.

**Requisito II: Estudio del funcionamiento e implementación del analizador.**

Debido a que la extensión Citus permite dos métodos de distribución, hash y append, es necesario conocer el código utilizado para el cálculo de cada una de las gráficas. Con el fín de identificar cuáles son los atributos de las tablas que se utilizan en cada una y así poder elegir el campo más adecuado para utilizarlo como columna de distribución.

**Requisito III: Evaluar si la paralelización mejora los tiempos de respuesta.**

Una vez realizado el diseño e integración de la base de datos paralelizada, es necesario evaluar si existe una mejora en los tiempos de cálculo de las gráficas. Para ello es necesario realizar pruebas en la configuración de Citus, como puede ser la modificación del número de Workers utilizado, la cantidad de fragmentos por tabla, los tiempos de consulta entre Master-Slave, etc.

**Requisito IV: Las soluciones de optimización deben integrarse correctamente.**

La correcta integración es el requisito más importante de todos, ya que de cara al usuario la aplicación simplemente debe ser más rápida, sin obligar a realizar configuraciones adicionales.

A continuación se presentan 3 figuras para ilustrar los objetivos de diseño de este TFG, tanto el punto desde el que se parte así como el funcionamiento que se espera conseguir con este proyecto.

En el primero de ellos se observa el funcionamiento inicial del analizador de trazas PrimeAnalytics. Como se puede ver, el usuario puede elegir entre cargar los 3 archivos que necesita la herramienta (fichero S11, fichero de trazas y fichero de topología), los cuales son procesados por la herramienta y almacenados en base de datos, o bien puede elegir visualizar los datos.

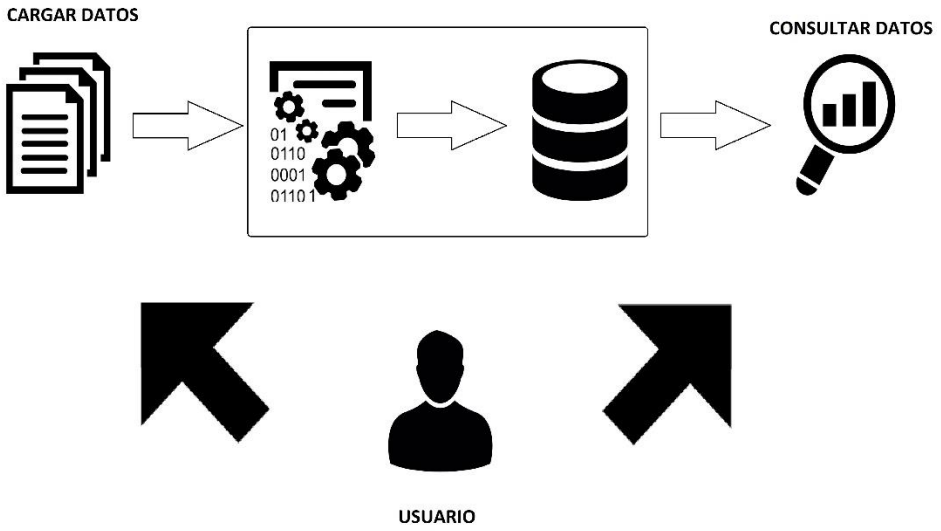



*Figura 16: Esquemático del funcionamiento original de PrimeAnalytics*

En la segunda figura se muestra la opción de paralelización, como se puede observar, tanto las opciones, como lo que percibe el usuario no se ve afectado. Esta modificación reside en la utilización de Citus el cuál define un nodo Master que se encarga de repartir los datos procesados a través de los Workers que forman el clúster, de esta forma se persigue reducir los tiempos de respuesta la aplicación mediante el uso de consultas en paralelo.

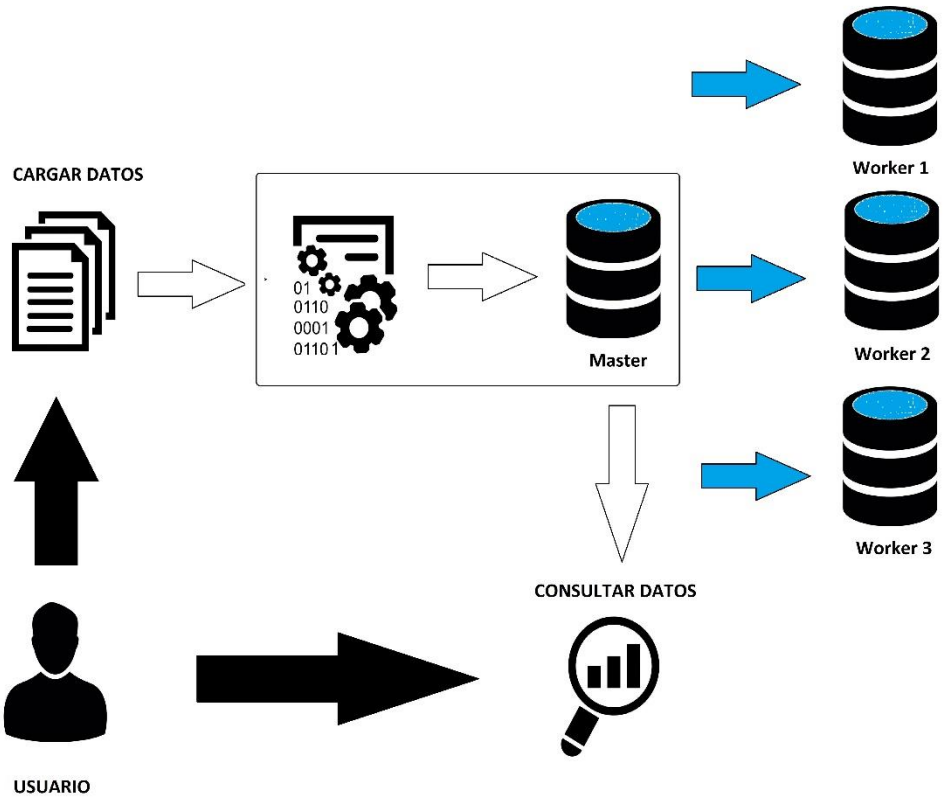



*Figura 17: Esquemático del funcionamiento PrimeAnalytics con BBDD distribuida*

Por último, en la tercera figura se expone la opción de computación en la nube. En ella se hace uso de los recursos ofrecidos por el Cloud para alojar la aplicación PrimeAnalytics, como puede observarse el usuario no percibe ningún tipo de cambio ni se ve obligado a hacer uso de la herramienta de forma diferente al encontrarse en la nube. Con esta implementación se persigue optimizar los tiempos de respuesta haciendo uso de los recursos disponibles en AWS, así como la evaluar uso de recursos cloud vs los recursos tradicionales.

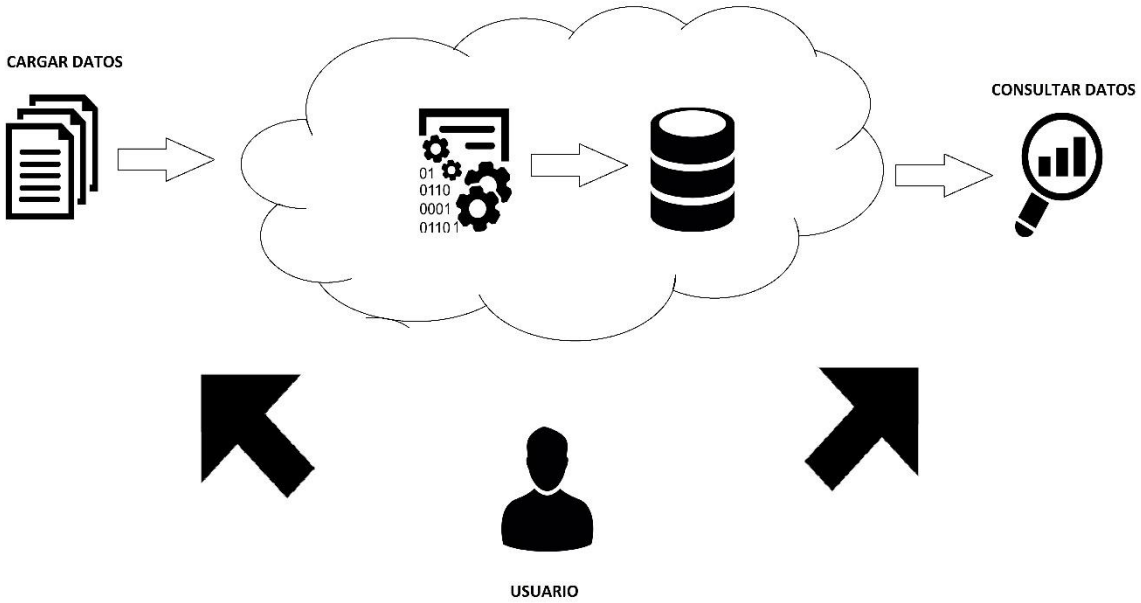



*Figura 18: Esquemático del funcionamiento PrimeAnalytics desplegado en el Cloud*

## 3.2. Selección de tecnologías

Como se concluyó en el capítulo 2, para el desarrollo de este Trabajo de Fin de Grado se utilizan diversas tecnologías, las cuales se engloban dentro de dos grupos.

El primero es el conjunto de herramientas utilizadas para desarrollar el analizador, entre las que destacan Docker, Django, PostgreSQL y Python. Estas tecnologías son fijas, ya que la herramienta hace uso de ellas, lo cual han servido de condicionante a la hora de elegir el resto de tecnologías utilizadas.

El segundo grupos está formado por tecnologías utilizadas para la realización de la paralelización y la computación en la nube.

En lo referente a la paralelización, inicialmente se comenzó a investigar sobre la herramienta PgPool, la cual permitía la replicación de tablas para aumentar la disponibilidad y el uso de un clúster de nodos a través de los cuales se reparten las tablas fragmentadas. Tras

intentar realizar la configuración, se vio que la fragmentación de tablas para realizar consultas en paralelo estaba en desuso en las versiones más recientes.

Finalmente se optó finalmente por la utilización de la extensión de PostgreSQL denominada Citus, debido a que se basa en la antigua extensión PgShard, posee varios canales de comunicación en los cuales se aportan soluciones e informan de nuevas mejoras (Stackoverflow y Google Groups) se encuentra bien documentada en su sitio web [Citus data] y se podía integrar en la herramienta mediante el uso de la imagen oficial de Citus para Docker.

En cuanto a la computación en la nube, tras realizar el estudio de las principales Clouds disponibles:

- FIWARE
- Microsoft Azure
- IBM SoftLayer
- Google Cloud Platform
- Amazon Web Services

Se optó por AWS debido a que se trata de una plataforma consolidad, con experiencia en el sector que ofrece una comunidad de desarrolladores, foros de ayuda, ejemplos de uso y una gran documentación a través de su sitio web oficial.

Otro de los motivos importantes a la hora de elegir esta plataforma frente a sus competidoras fue su cuenta gratuita de prueba, la cual supera en términos temporales al resto, este era un punto decisivo debido a que la curva de aprendizaje de este proyecto era bastante grande y podía suceder que se excediera el tiempo ofrecido en el resto de cuentas de prueba.

## 3.3. Etapas de diseño

Para desarrollar correctamente el proyecto se fijaron una serie de etapas en esta fase de diseño, las cuales se exponen a continuación:

1. Estudio del funcionamiento interno de PrimeAnalytics
2. Manejo y aprendizaje del uso de las herramientas Docker, Django y PostgreSQL
3. Aprendizaje del lenguaje de programación Python
4. Utilización de la extensión Citus para PostgreSQL y realización de pruebas
5. Integración de Citus en PrimeAnalytics
6. Migración de PrimeAnalytics a AWS
7. Pruebas y evaluación de las técnicas de optimización usadas.

# Capítulo 4. Desarrollo e integración

En este capítulo se detalla la implementación de las técnicas de optimización elegidas siguiendo los principios y metodología de diseño establecidos en el capítulo 3. Se comenzará explicando las fases llevadas a cabo para desarrollar la paralelización de la base de datos, las decisiones tomadas durante el desarrollo, la problemática encontrada así como las diversas modificaciones realizadas en el analizador. A continuación, debido a las ventajas ofrecidas por la paralelización, se detalla la implementación de una nueva funcionalidad para la herramienta. Para finalizar se detallará el proceso de migración a la nube realizado.

## 4.1. Elección del método de distribución

A la hora de desarrollar la opción de paralelización de la base de datos PostgreSQL en la herramienta de análisis de trazas se podía que elegir entre 2 métodos de distribución, hash y append. Como se ha comentado en apartados anteriores, cada uno de estos métodos está enfocado a casos distintos, y optimizan la búsqueda de determinados datos.

**Método I: Append.**

Para datos ordenados en el tiempo, se consigue una mejora a la hora de realizar consultas que impliquen usar el atributo de tabla utilizado como columna de distribución. En este caso, puesto que los datos, tanto de nodos como de trazas, llegan ordenados temporalmente se ha elegido el atributo "*pctime*" como columna de distribución.

De esta forma, al utilizar el campo pctime conseguimos que los datos se repartan temporalmente en los fragmentos de los Workers. Concretamente, a la hora de cargar los datos en el analizador de trazas, se proporcionan 3 archivos:

- Fichero csv con las trazas capturadas
- Fichero csv con la evolución de la topología
- Fichero xml, denominado S11

El fichero S11 contiene información sobre la topología de la red en el tiempo en el que se ha capturado, debido a esto, todos los datos del S11 comparten la misma fecha. Por el contrario, tanto el fichero de trazas como el de evolución, contienen datos con fechas diferentes pero ordenados temporalmente.

Antes de comenzar con la carga de estos 3 ficheros se ha de configurar la base de datos paralelizada, se deben introducir una serie de comandos para configurar la extensión de Citus. Para realizarlo de forma automática se ha usado un script denominado "*citus_init.py*" que hace uso de la extensión de Python, psycopg2, a través de la cual se han enviado a la base de datos los siguientes comandos:

```
database = psycopg2.connect(database="primeanalytics",user="primeanalytics",password="1234",host="citus_master",port="5432")
database.autocommit = True
cursor = database.cursor()

cursor.execute("ALTER TABLE public.trace_parser_app_node DROP CONSTRAINT trace_parser_app_node_pkey")
database.commit()
cursor.execute("ALTER TABLE public.trace_parser_app_traces DROP CONSTRAINT trace_parser_app_traces_pkey")
database.commit()
cursor.execute("SELECT master_create_distributed_table('trace_parser_app_traces','pctime','append')")
database.commit()
cursor.execute("SELECT master_create_distributed_table('trace_parser_app_node','pctime','append')")
database.commit()
```

*Figura 19: Configuración inicial Citus*

En primer lugar se realiza la conexión con la base de datos utilizando el DNS del Worker y definiendo nombre de la BBDD, usuario, puerto y contraseña.

Seguidamente, se han de eliminar las Primary Key ya que la columna de distribución ha de coincidir con el Primary Key en caso de estar definidas. Este paso se debe realizar ya que Django establece las Primary Keys al inicio, tanto sobre la tabla "*trace_parser_app_node*" que se utiliza para los datos de nodos como para "*trace_parser_app_traces*" que contendrá la información sobre las trazas, más adelante veremos cómo se ha solucionado el tema del borrado de las PKs.

Posteriormente se utiliza una función propia de Citus que se puede ver en la Figura 16, "*master_create_distributed_table*", la cual tiene como parámetros de entrada la tabla sobre la cual se va a realizar la distribución, el campo o columna de distribución y el método de distribución. A partir de aquí nuestra base está configurada para comenzar a utilizarla como base de datos distribuida.

Tras esta configuración inicial, se han de realizar una serie de acciones que tienen lugar en el fichero "*functions.py*". Este fichero es el que utiliza la herramienta tanto para cargar los 3 ficheros de entrada como para calcular las gráficas. Los siguientes pasos se aplican de forma idéntica para los 3 archivos, con la salvedad de que en cada uno se utiliza la tabla (nodos o trazas) correspondiente.

**Paso I: Definición de tablas temporales**.

Una de las modificaciones requeridas debido al uso de Citus, es la creación de tablas temporales en Django. Esto es necesario ya que Citus crea un fragmento vacío en la tabla distribuida, y agrega a ese fragmento la tabla definida como temporal, que es la que contiene los datos.

La tabla temporal se define como un modelo de datos en Django, exactamente igual a la tabla original, no se hace ninguna modificación sobre ella, por lo que contiene las Primary Keys.

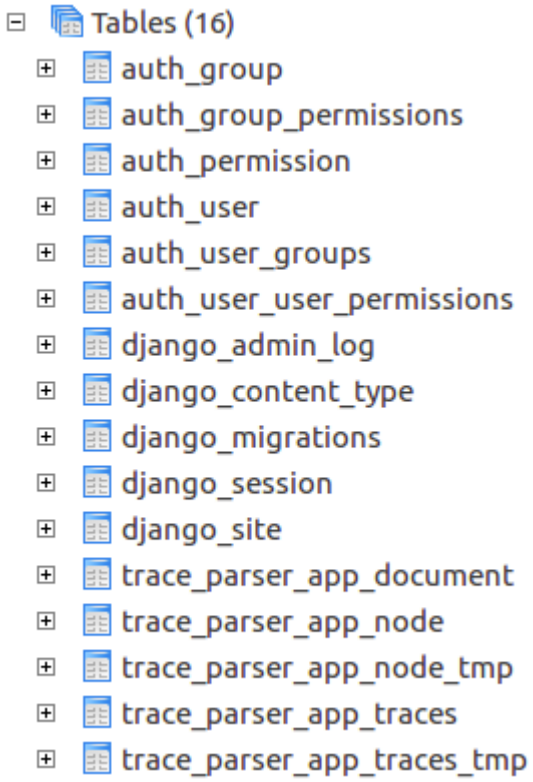



*Figura 20: Tablas temporales*

**Paso II: Inserción de datos en la tabla temporal**.

Cuando se cargan los 3 ficheros en la herramienta, se agregan los datos a las tablas temporales mediante un comando ORM [49] proporcionado por Django, denominado "*bulk create*". Este comando permite la inserción de un array de objetos a la base de datos, acelerando considerablemente la inserción de los mismos. Este es uno de los principales motivos por el cual se eligió append para toda la herramienta, el uso de las tablas temporales no distribuidas permiten usar comandos ORM al contrario que las distribuidas, que actualmente no lo soportan.

**Paso III: Ajuste del factor de replicación**.

En este paso simplemente se define el número de réplicas que queremos de ese fragmento, en este proyecto se ha forzado a 1.

**Paso IV: Agregar tabla a un fragmento**.

Para conseguir agregar los datos finalmente a la tabla distribuida se hace uso de la función de Citus "*master_append_table_to_shard*" y de la función "*master_create_empty_shard*", la siguiente línea servirá para ilustrar el proceso:

```
master_append_table_to_shard(master_create_empty_shard('trace_parser_app_traces'),'trace_parser_app_traces_tmp','citus_master',5432)
```

En la primera función se definen un fragmento vacío, la tabla que va a ser copiada a ese fragmento, el DNS del nodo donde se ha de copiar y el puerto. La segunda función se encarga de crear un fragmento vacío en la tabla distribuida.

**Paso V: Eliminación de los objetos de las tablas temporales.**

Una vez agregado correctamente los datos se vacía la tabla temporal para poder ser utilizada de nuevo.

Uno de los criterios de diseño para la inserción de los datos en los fragmentos, es que cada uno almacene únicamente datos capturados en el mismo día, utilizando para ello el atributo "*pctime*", no puede haber en un fragmento datos de dos días diferentes. Para ello se hace uso de un bucle que se encarga de ir agregando al mismo fragmento los datos siempre que sean del mismo día, de lo contrario crea un fragmento nuevo.

Hay que destacar que cada vez que se carga el paquete de 3 archivos los datos se agregan en fragmentos separados debido a la definición del script "functions.py". Esto es un condicionante que viene impuesto por la forma que tienen los concentradores PRIME de capturar los datos, como S11 se capturan de forma periódica, cada vez que se ha capturado un S11, se ha de subir individualmente junto a los dos archivos (trazas y nodos) que se correspondan con él.

**Método II: Hash.**

Este método es muy útil para realizar búsquedas rápidas en tablas con una elevada cantidad de datos. Se basa en el método de búsqueda por clave-valor y como se ha explicado anteriormente, es necesario definir un atributo de la tabla como columna de distribución.

El principal problema de utilizar el método hash en la aplicación de análisis de trazas es que dicha aplicación maneja dos tablas, una para la información de los nodos y otra para la información de las trazas. Como se detalla a continuación, los campos en ambas tablas son muy diferentes:

| Trace_parser_app_node | Trace_parser_app_traces | |
|---|---|---|
| id | id | lnid |
| scenario | scenario | prio |
| file_path | file_path | length |
| pctime | pctime | crc |
| file_type | time | txt |
| mac | htype | scheme |
| state | do | phylevel |
| level | level | physnr |
| switch_mac | hcs | pna |
| signa | sid | specific_type |

*Tabla 3: Atributos de las tablas nodo y traza*

Observando ambas tablas se puede ver que tan solo comparten 4 atributos (id, scenario, file_path y pctime). Por lo que habría que utilizar uno de ellos como columna de distribución, esta obligación en el diseño plantea dos opciones:

La primera de ellos es que Django establece como Primary Key el campo ID por lo que se debería utilizar como columna de distribución. El problema es que las consultas a la base de datos que hacen uso de la tabla de nodos no utilizan ese campo en sus definiciones por lo que no se aprovecharía la ventaja del uso de hash.

La segunda sería utilizar cualquiera de los otros 3 atributos, pero como se verá más adelante las consultas que utilizan la tabla de nodos usan atributos diferentes.

Después de observar tanto las definiciones de las consultas como las de las tablas se llegó a la conclusión de que no había un campo común en todas que facilitara el uso de hash. El campo que más se aproximaba era el campo pctime, pero la búsqueda en intervalo temporal se ralentizaba al utilizarlo.

No obstante, en el capítulo de validación y resultados se verá el rendimiento hash aplicado a gráficas de forma individual.

## 4.2. Justificación de la elección

Pese a que el método de distribución hash es más rápido en búsqueda, debido al sistema de clave-valor, que el método append, finalmente se optó por utilizar append en las dos tablas por los siguientes motivos.

Como se ha explicado arriba, a la hora de configurar la tabla para utilizar hash hay que definir la columna de distribución. El mayor problema es que el Primary Key que establece Django por defecto es el campo "ID", por lo que obligaba a tomar dos opciones.

La primera de ellas era eliminar el Primary Key y así poder utilizar cualquier campo de la tabla como columna de distribución, pero esto acarrearía problemas cuando los datos aumenten, ya que podría darse el caso de duplicar o re escribir datos por error ya que no habría un campo inequívoco para poder diferenciarlos.

La segunda opción era utilizar el "*ID*" como columna de distribución Hash, evitando así problemas de sobre escritura, pero en las pruebas de desarrollo se vio que ninguna consulta del analizador utilizaba el campo "*ID*" para obtener resultados, por lo que no se apreciaba ninguna mejora en el rendimiento de las mismas.

Por ello se realizó el diseño, adaptación e integración de la base de datos distribuida basada en el método de distribución append.

## 4.3. Modificación del docker-compose.yml

Antes de nada, para comenzar a utilizar Citus se deben modificar una serie de archivos que permiten la inicialización de la aplicación web del analizador de trazas. Como se habló en capítulos anteriores, esta herramienta está desarrollada sobre Django y Docker por lo que para integrar la extensión de PostgreSQL que facilita la paralelización de la misma es necesario modificar el archivo "*docker-compose.yml*".

En la versión normal de la herramienta existían únicamente dos servicios, el primero de ellos era la base de datos la cual hacía uso de la imagen PostgreSQL facilitada por Docker. El segundo es la propia herramienta, PrimeAnalytics, la cual se basaba en una imagen de Django.

```
services:
  db:
    image: postgres
    environment:
          POSTGRES_USER: primeanalytics
          POSTGRES_PASSWORD: SKSBn94GYnC26
          POSTGRES_DB: primeanalytics

  prime_analytics:
    image: django
    environment:
          DATABASE_USER: primeanalytics
          DATABASE_PASSWORD: doSKSBn94GYnC26
          DATABASE_HOST: db
          DATABASE_PORT: 5432
          DATABASE_NAME: primeanalytics
          UI_USER: admin
          UI_PASSWORD: primeanalytics
    ports:
      - "8000:8000"
```

*Figura 21: Fichero docker-compose.yml inicial*

Como puede verse en la figura, ambos servicios tienen declaradas variables de entorno como el usuario y contraseña de la base de datos, nombre de la base de datos, usuario y contraseña de la aplicación y los puertos, estas variables son necesarias para la configuración inicial. Además, este fichero llama al entrypoint, "*docker-entrypoint.sh",* que contiene una serie de comandos utilizados en el proceso de inicio de la herramienta, más adelante veremos que este .sh también ha tenido que ser modificado para manejar correctamente Citus.

En la versión que incluye la paralelización se han agregado 3 servicios adicionales:

- Master: se trata del nodo que se encargará de recibir las peticiones del analizador y paralelizar las consultas hacia el clúster de nodos Workers, donde se encuentran realmente los datos.
- Worker: son los nodos que forman el clúster en los cuales se guardaran y se buscarán los datos.
- Config: es el servicio encargado de mantener la lista de Workers que forman el clúster e informar de ello al Master.

En la figura del nuevo docker-compose.yml se ve como los servicios master y Worker se basan ambos en la imagen de Citus, en concreto en la versión 5.2.0.

```
services:
  master:
    container_name: 'citus_master'
    image: 'citusdata/citus:5.2.0'
    environment:
          POSTGRES_USER: primeanalytics
          POSTGRES_DB: primeanalytics
    ports: ['5432:5432']
    labels: ['com.citusdata.role=Master']


  worker:
    image: 'citusdata/citus:5.2.0'
    environment:
          POSTGRES_USER: primeanalytics
          POSTGRES_DB: primeanalytics
          DATABASE_USER: primeanalytics
          DATABASE_PASSWORD: SKSBn94GYnC26
    labels: ['com.citusdata.role=Worker']


  config:
    container_name: 'citus_config'
    image: 'citusdata/workerlist-gen:0.9.0'
    volumes: ['/var/run/docker.sock:/tmp/docker.sock']
    volumes_from: ['master']

  prime_analytics:
    image: django
    environment:
          DATABASE_USER: primeanalytics
          DATABASE_PASSWORD: SKSBn94GYnC26
          DATABASE_HOST: master
          DATABASE_PORT: 5432
          DATABASE_NAME: primeanalytics
          UI_USER: admin
          UI_PASSWORD: primeanalytics
```

*Figura 22: fichero docker-compose.yml con Citus*

## 4.4. Modificación del entrypoint.sh

Una vez modificado el Docker Compose, es necesario modificar el fichero entrypoint.sh, el cual se encarga de ejecutar una serie de archivos proporcionados por Django para poner en marcha la aplicación. Si se ejecutara el docker-compose simplemente con las modificaciones anteriores, Django crearía las bases de datos no distribuidas por defecto, lo cual sería un inconveniente pues habría que modificarlas posteriormente de forma manual. Por ello se desarrolló un script en Python denominado "*citus_init.py*", adjuntado en los anexos, y ejecutado en el entrypoint. Este script elimina las Primary Key por el motivo descrito anteriormente, y utiliza la función proporcionada por Citus para distribuir las tablas de nodos y trazas.

Al incluir el script *citus_init.py* en el entrypoint, una vez Django inicie la aplicación, está tendrá las tablas distribuidas inicializadas.

## 4.5. Creación del clúster

Llegados a este punto, y una vez inicializada la aplicación, podemos hacer uso de la funcionalidad de escalado que proporciona Docker Compose. Mediante el comando:

***docker-compose scale worker=(num)***

Podemos crear el número de Workers que deseemos dentro de nuestro clúster, y automáticamente se registrará en el servicio de configuración para informar al Master de las direcciones IP en las que se encuentran los Workers que se acaban de agregar.

Se toma la información establecida para el servicio Worker en el fichero *docker-compose.yml*, de forma que estos nodos tienen una estructura muy similar al Master, visualizando todo el proceso desde el gestor de base de datos PgAdmin3.

Debido a que Django, al iniciar la aplicación, genera unos campos en las tablas denominados "NextVAL" encargados de incrementar el ID de cada dato cada vez que se inserta uno nuevo en la base, es necesario que los Workers tengan las secuencias adecuadas para realizar bien la gestión de los IDs.

Por defecto, estas secuencias solo se crean en el Master, para poder agregarlas a los Workers antes de comenzar a insertar datos se desarrolló el script "*scale.py*" adjunto en los anexos. Este script sustituye la necesidad de lanzar el comando de escalado de docker-compose, ya que se puede modificar una variable global que indica el número de Workers que se van a desplegar. El funcionamiento es el siguiente:

1. Se define la variable con el número de Workers a los que se va a escalar.
2. Se lanza un comando de terminal que contiene la orden de escalado.
3. Se ejecuta una línea que obtiene las IPs de todos los Workers desplegados.
4. Finalmente se conecta a cada uno de los Workers y se carga en ellos un archivo .sql que contiene las secuencias.

```
mario@mario-Inspiron-13-7359:~/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0
/prime-analytics$ python scale.py
Desired container number already achieved
Creating and starting primeanalytics_worker_2 ... done
Creating and starting primeanalytics_worker_3 ... done
Creating and starting primeanalytics_worker_4 ... done
('----------> Escalado a ', 4, ' Workers')
('Secuencias agregadas a la IP ', '172.18.0.2')
('Secuencias agregadas a la IP ', '172.18.0.6')
('Secuencias agregadas a la IP ', '172.18.0.7')
('Secuencias agregadas a la IP ', '172.18.0.8')
----------> Completado
```

*Figura 23: Ejecución de scale.py*

Una vez ejecutado el archivo *scale.py* con el número de Workers establecido la aplicación está lista para recibir los .csv y .xml que contienen los datos, en la siguiente figura se puede observar, utilizando la herramienta pgadmin3, los nodos que forman el clúster.

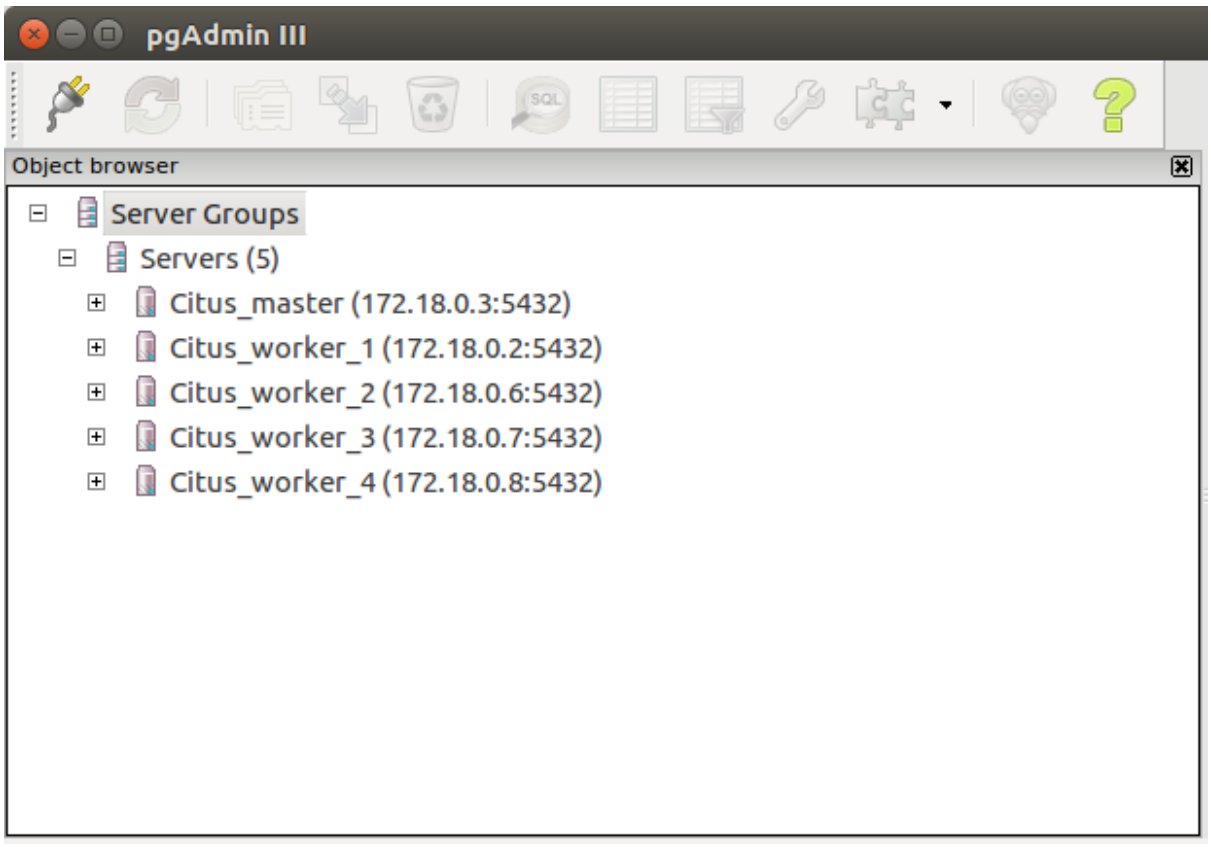



*Figura 24: Visualización de elementos del Cluster*

Por último, al cargar los ficheros en la aplicación se van insertando los datos en los Workers del clúster. En la siguiente figura se puede observar que los datos se almacenan en dos fragmentos de la tabla ubicados en el Worker número 1.

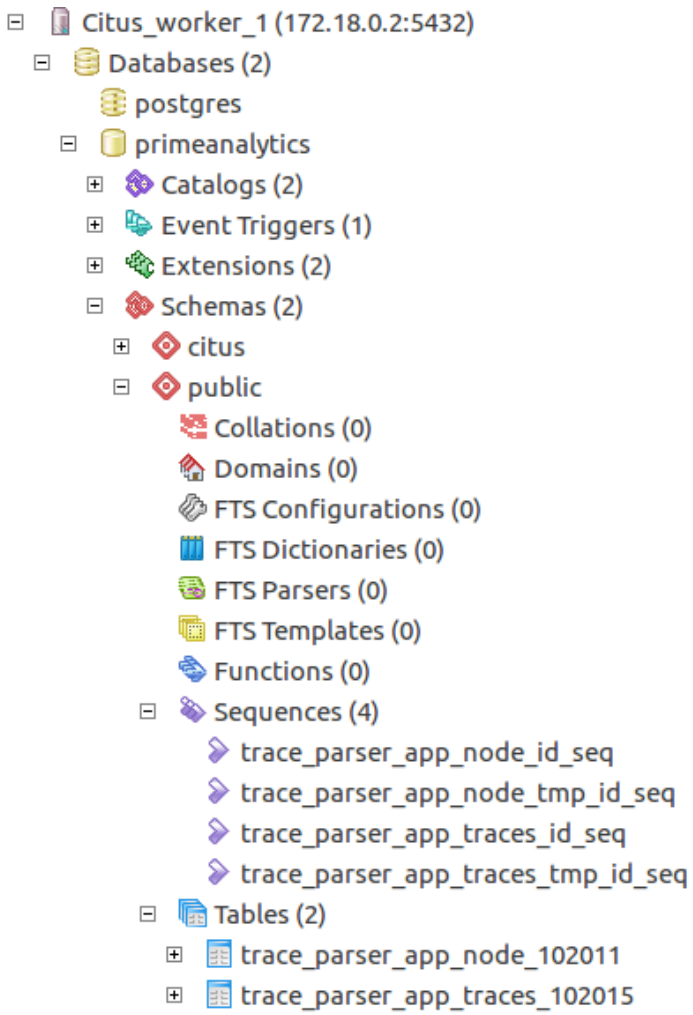



*Figura 25: Visualización de fragmentos en Worker*

## 4.6. Adaptación del gestor de archivos

La aplicación contaba inicialmente con un gestor de archivos a través del cual se pueden ver los ficheros que se han ido agregando a la base de datos. Permite aparte de la visualización, eliminar ficheros que se hayan podido subir por equivocación, ficheros repetidos o simplemente se desee vaciar completamente la base de datos.


*Figura 26: Gestor de Archivos*

Cada vez que se carga en la herramienta el paquete de tres archivos (trazas, topología y S11), Django guarda la información de esa subida en una tabla llamada “documents”, y asigna un ID para cada paquete de archivos.

En un funcionamiento normal, basta con saber el nombre del archivo, tomar su ID y borrarlo, pero al utilizar una base de datos distribuida es necesario eliminar datos en diferentes fragmentos, por lo cual hay que saber en qué fragmentos se encuentran los datos correspondientes al archivo que se quiere eliminar. La forma que se ha optado para que funcione correctamente con la BBDD distribuida es obtener el intervalo temporal de los archivos, de forma que no es necesario que todos los datos estén en un único fragmento.

Para poder mantener esta funcionalidad de la herramienta se ha modificado el archivo “*views.py*”, en el cual está programado el gestor, se han realizado las siguientes modificaciones:

1. Se filtra por el ID seleccionado a través del gestor, y se obtienen los campos, trace_file, s11_file y topology_file. Estos campos hacen referencia al nombre que se asigna al archivo cuando se sube a la aplicación.

2. Una vez obtenido los nombres, se filtra en las tablas de nodo y trazas por el ID, obteniendo el atributo “*pctime*” de cada uno.

3. Se calcula el tiempo inicial y final que aparece en los archivos de trazas y topologías, el tiempo del archivo S11 es el mismo para todos sus datos.

4. Tras obtener el intervalo temporal al que pertenecen los archivos se lanza una consulta para eliminar todos los archivos que pertenezcan a ese intervalo. Utilizando la función de Citus:

   *SELECT * FROM master_apply_delete_command ()*

5. Una vez eliminados todos los datos, se ha de eliminar de la tabla documents el ID que corresponde a los archivos.

## 4.7. Problemática encontrada y soluciones adoptadas.

Durante el desarrollo de este proyecto han surgido una serie de problemas a la hora de decidir cómo utilizar Citus y cómo integrarlo con la herramienta. A continuación se exponen los problemas más importantes a los que fue necesario que enfrentarse durante las fases de desarrollo e integración.

**Problema 1: El uso de dos tablas diferentes por parte del analizador**

El primer problema que surgió en la fase de desarrollo fue que el analizador utiliza dos tablas diferentes para generar las gráficas. Esto no presentaba un gran problema para el desarrollo en sí de la paralelización, pero sí suponía un problema para elegir los atributos óptimos para utilizarlos como columna de distribución.

Se analizaron una por una las consultas que realiza el analizador para cada gráfica, identificando los atributos en común, y viendo en cuales un aumento de la velocidad de búsqueda supondría una mejora para la aplicación.

**Solución adoptada:**

Tras realizar pruebas y comprobar las definiciones del código se decidió utilizar el atributo pctime como columna de distribución del método append. Debido principalmente a que los ficheros proporcionados por las distribuidoras contenían los datos en orden temporal.

**Problema 2: Variabilidad de los atributos**

Este problema tiene relación con el anterior, en este caso la escasa variabilidad de ciertos atributos, que en un primer momento eran candidatos para utilizarse como columnas de distribución, pero que posteriormente fueron descartados. Tras una serie de desarrollos fallidos contacté con el soporte de Citus y me informaron que los atributos idóneos para utilizar son aquellos que tienen una gran variabilidad, de esta forma es más fácil repartir de forma uniforme los datos entre los fragmentos, evitando así que algunos queden muy cargados y otros prácticamente vacíos.

**Solución adoptada:**

Después de conocer esa información las opciones se reducían al uso del ID o el pctime, se acabó eligiendo el segundo porque era un campo común en todas las consultas.

**Problema 3: Elección de compromiso entre los métodos de distribución**

Al iniciar las pruebas de desarrollo tanto con el método hash como el append antes de decidir cuál de los dos utilizar se vio que con hash había ciertas consultas en las cuales se

conseguía reducir el tiempo drásticamente. El método append conseguía una reducción importante de tiempo en varios escenarios de prueba, pero era más lento que el hash. Era necesario tomar la decisión sobre que método de distribución utilizar.

**Solución adoptada:**

Finalmente, como se ha descrito al inicio de esta sección, debido a que el método hash era el más complicado de integrar y de que solo optimizaría una o dos gráficas empeorando los tiempos de respuesta del resto, se optó por utilizar append, ya que en la solución de compromiso era la que mayor optimización conseguía.

**Problema 4: Comandos ORM**

Este problema se presentó cuando Citus se encontraba en su versión 5.1.0, el cual no permitía utilizar estos comandos en bases de datos distribuidas, se buscó alguna solución alternativa pero finalmente con el lanzamiento de la versión 5.2.0 en Julio de 2016 se solucionó.

**Problema 5: Integración en Django**

Debido a que Django crea al inicio la base de datos con las tablas definidas, y que uno de los requisitos es evitar que el usuario realice configuraciones adicionales, es necesario que en la puesta en marcha se ejecuten los comandos de configuración de Citus. A demás se encuentra el problema de no poder realizar inserción de los datos mediante comandos ORM directamente sobre tablas distribuidas.

**Solución adoptada:**

Se utilizó la extensión de Python psycopg2 para conectarse directamente a la base de datos en la fase de inicialización para ejecutar los comandos necesarios. En lo referente a la inserción de datos se crearon unas tablas temporales definidas como modelos Django para conseguir un funcionamiento correcto.

## 4.8. Desarrollo de módulo software para la evolución de la topología

Durante el desarrollo e investigación de las soluciones de optimización para la herramienta de análisis de trazas PrimeAnalytics, se desarrolló un módulo software programado en Python e integrado en el framework Django que permite obtener una visión jerárquica de la topología en un instante determinado de la red de contadores PRIME. Este módulo es de especial importancia ya que permite evaluar el alcance de determinados problemas y su repercusión en la red. El motivo del desarrollo de este módulo, aparte de aportar valor a la herramienta PrimeAnalytics, es que como se verá a continuación, se trata de

un módulo que genera una cantidad muy elevada de nuevos datos los cuales se pueden procesar de forma más eficiente al utilizar una base de datos paralelizada.

A continuación se explica brevemente el objetivo, desarrollo y resultado de este módulo Software.

**Versión 1 y funcionamiento básico**:

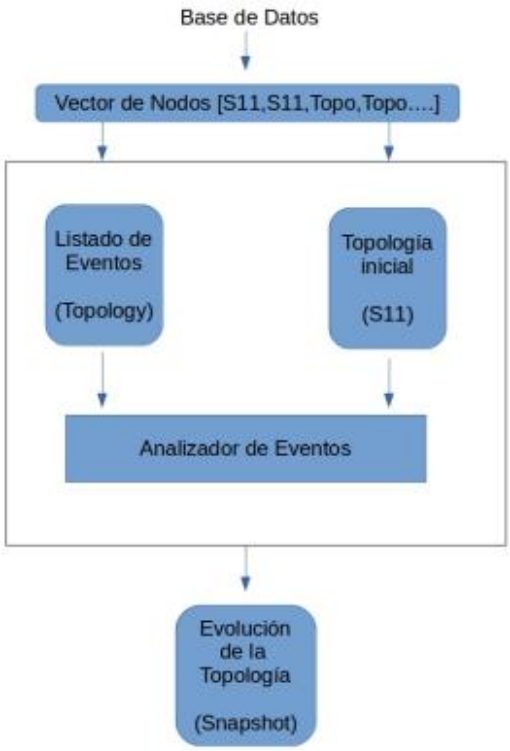



*Figura 27: Funcionamiento del módulo Software*

Con este módulo se pretende obtener una visión lógica del estado de la topología de red en función de los cambios que se vayan realizando en ella. Inicialmente se parte de una topología capturada por el concentrador en un instante determinado, fichero S11, y tiene asociada una serie de eventos de topología.

Una vez analizados todos los eventos de topología se obtendrían "imágenes" denominadas Snapshot que permiten ver la topología en un instante de tiempo determinado.

Inicialmente el módulo toma 3 atributos de entrada, un vector de objetos Nodo, una fecha de inicio y una fecha de fin. El objetivo de utilizar fecha de inicio y de final es poder acotar la lista de eventos en un intervalo temporal, para evitar analizar todo el listado completo de eventos.

Una vez recibido el vector de objetos Nodo, se realiza un primer filtrado por tipo, ya que en ese vector se encuentra nodos "S11" y nodos "TOPOLOGY".

Los S11 se corresponden al estado inicial de la red en un momento determinado, y los Topology se corresponden con los eventos que se han ido produciendo sobre la topología a lo largo del tiempo de captura de esta información.

Una vez realizado el pre-filtrado del vector inicial, se obtienen dos vectores, uno contendrá únicamente los nodos con el campo "file_type" igual a S11 y otro con ese mismo

campo igual a Topology. A partir de aquí, se analiza evento por evento para modificar correctamente el S11.

La función está implementada de forma que se realiza un copia de la lista de nodos S11, denominada tmp (temporal), para realizar las modificaciones sobre ella y conservar intacta la topología inicial. Debido a esto las modificaciones de los eventos se producirán sobre el vector tmp.

El resultado de esta función es un vector de capturas o snapshot, que contendrá el vector tmp modificado según el tipo de evento que se haya procesado. Para conseguir reducir el volumen de información, pero sin perder detalle de las constantes modificaciones, se ha optado por lanzar capturas en un tiempo determinado (variable según el usuario).

Básicamente se analiza evento por evento y se modifica el vector tmp, una vez alcanzado el tiempo fijado para este procesado se lanza la captura o Snapshot. Para poder obtener dicho funcionamiento se han creado dos variables, “delta_time” y “snapshot”.

Delta_time es una variable objeto con la que se puede variar el tiempo con el que se lanzan las capturas, p.ej. si está fijada a 60 segundos, la función empezará a analizar eventos, y una vez pasado el tiempo definido por delta_time, se añadirá el vector tmp, que contiene todas las modificaciones hasta el momento, al vector snapshot.

Snapshot se trata de un vector en el que guardaremos los vectores tmp cada vez que se lance una captura al llegar al valor temporal definido por delta_time. Hay que destacar que todos los nodos contenidos en los vectores tmp que se agregan al vector Snapshot tienen modificado su campo file_type a SNAPSHOT para diferenciarlos de los tipos comentados anteriormente (S11 y Topology).

Atributos de los objetos Node:

El vector de entrada estará formado por objetos Node extraídos de la base de datos, estos objetos Node tienen los siguientes campos:

- Escenario (scenario)
- Archivo de la traza (trace_file)
- Tipo de archivo (trace_type)
- Tiempo (pctime)
- Dirección Mac propia (mac)
- Estado (state)
- Nivel (level)
- Switch Mac (switch_mac)
- Nivel de SNR (signal_level)

El atributo “state”, puede tener los siguientes estados:

- ❖ 0 Desconectado
- ❖ 1 Conectado (Terminal)
- ❖ 2 Switch

El atributo file_type puede ser:

- ❖ S11
- ❖ TOPOLOGY
- ❖ SNAPSHOT

Funcionamiento del procesado según los tipos de evento:

Para procesar evento por evento, se recorre la lista de eventos uno por uno evaluando en primer lugar si la dirección mac del nodo del evento se encuentra en la lista de la topología válida hasta ese momento.

Para ello se toma la mac del evento y se realiza la comprobación, pueden darse dos casos, que la mac se encuentre en la topología, o que por el contrario no se encuentre en ella. Si la mac es encontrada, se evalúa el campo 'state' del evento, el cuál indicará el estado del nodo (Disconnect (0), Connect (1) o Switch (2)):

- State = 0, indica que el nodo se ha desconectado, por lo que se deben realizar los siguientes cambios:

  - ➔ Actualizar el estado
  - ➔ Poner el campo de nivel a 0
  - ➔ Poner la switch_mac a Null
  - ➔ Modificar el file_type a SNAPSHOT

- State = 1, indica que el nodo se ha conectado, al encontrarse en la topología anterior (aunque en estado desconectado o como switch), se deben realizan los siguientes cambios:

  - ➔ Actualizar el estado
  - ➔ Poner la switch_mac que diga el evento
  - ➔ Actualizar el campo de nivel según el nivel del padre al que se conecte
  - ➔ Modificar el file_type a SNAPSHOT

- State = 2, indica que el nodo a pasado a ser switch, por lo que se deben realizar los siguientes cambios:

- ➔ Actualizar el estado
- ➔ Poner la switch_mac que diga el evento
- ➔ Actualizar el cambio de nivel según el nivel del padre al que se conecte
- ➔ Modificar el file_type a SNAPSHOT

En este caso no se añade ningún nodo nuevo al vector tmp, pero si hay que recorrer todos los elementos y cambiar el campo file_type de todos a SNAPSHOT y cambiar la fecha por la fecha del evento.

Si una vez recorrida la lista de topología no se encuentra un nodo cuya mac se corresponda a la mac del evento, se crea un nuevo nodo que se agrega al vector tmp. Para ello se define un nuevo objeto Nodo cuyos atributos serán los atributos que aparezcan en el evento.

De igual manera, una vez añadido al vector tmp, se modifica el campo 'file_type' y 'pctime' de todos los nodos que se encuentran en el vector tmp.

Generación de la captura de topología:

Como he comentado anteriormente, la función va procesando evento por evento y modificando el vector tmp en función de los diferentes eventos. Pero a su vez se va teniendo el cuenta el tiempo que transcurre entre los eventos para poder lanzar la captura en función del valor de la variable delta_time.

Para poder realizar esta parte, se toma el tiempo del primer evento como referencia y se observa la diferencia de tiempo que hay con el siguiente evento. La captura se lanzará cuando la diferencia entre el tiempo de referencia (primer evento) y el evento/s posterior sea menor o igual que el delta_time.

Una vez que se lance la captura, que consiste en añadir el vector tmp al vector snapshot, el tiempo de referencia se actualiza al tiempo del evento que ha producido la captura y se vuelve a realizar el proceso de nuevo.

La función parará cuando se llegue al tiempo final del intervalo y devolverá el vector Snapshot.

**Versión 2 de la generación y captura de Snapshot (Intervalos Temporales)**

En esta segunda versión de capturas se ha implementado una nueva función complementaria a la existente (generate_snapshot). Esta función nueva (generate_interval) recibe un intervalo temporal fijado por el usuario a través de la interfaz gráfica del analizador.

Se ha partido de la base de que se pueden analizar archivos que contengan varios S11 o Snapshot que se tomarán como referencia a la hora de analizar y actualizar la red en función de los eventos de topología que se vayan recibiendo.

Inicialmente se realiza un filtrado sobre los eventos cuyo campo “file_type” sea igual a S11 o SNAPSHOT y se obtiene el tiempo de estos. Dado que en el instante temporal en el que se generan tanto los S11 como los SNAPSHOT se ven implicados varios contadores, al realizar el filtrado se obtendrán fechas repetidas, debido a esto es necesario una lista que contenga únicamente fechas y horas sin repetir.

Ej.

| Lista Original | | Lista Filtrada sin repeticiones | |
|---|---|---|---|
| S11 | 2016/2/01 10:10:13 | S11 | 2016/2/01 10:10:13 |
| S11 | 2016/2/01 10:10:13 | SNAP | 2016/2/01 10:14:12 |
| S11 | 2016/2/01 10:10:13 | S11 | 2016/2/01 10:30:21 |
| SNAP | 2016/2/01 10:14:12 | | |
| SNAP | 2016/2/01 10:14:12 | | |
| S11 | 2016/2/01 10:30:21 | | |
| S11 | 2016/2/01 10:30:21 | | |

Una vez obtenida la lista sin repeticiones, el funcionamiento es el siguiente:

1. Definimos dos vectores strart_time y end_time, que serán los parámetros que se pasarán a la función generate_snapshot

2. Se recorre la lista filtrada sin repeticiones (S11_periodo).

3. Se comprueba cada elemento de la lista con el tiempo inicial proporcionado por el usuario.

4. Si el tiempo es mayor o igual al proporcionado por el usuario se calcula la diferencia entre el evento actual y su siguiente. Si esa diferencia es mayor que 60 segundos, añade los tiempos a los vectores start_time y a end_time.

5. Hay que tener en cuenta la siguiente consideración, debemos añadir a start_time el evento anterior al que estamos. Para aclarar el porqué de esto se utilizará un ejemplo sencillo

Ej. Fecha inicio usuario 2016/2/01 10:10:13

| | | |
|---|---|---|
| S11 | 2016/2/01 | 10:10:10 |
| S11 | 2016/2/01 | 10:10:15 |
| S11 | 2016/2/01 | 10:10:20 |
| S11 | 2016/2/01 | 10:10:25 |

Considerando la fecha y hora fijada por el usuario, la primera fecha que se incluiría en el vector start_time ha de ser el S11 2016/2/01 10:10:10, ya que entre el 2016/2/01 10:10:10 y 2016/2/01 10:10:15 habrá eventos. Si agregamos como primer tiempo 2016/2/01 10:10:15, no estamos dando una información real, ya que entre 10:10:13 y 10:10:15 pueden suceder eventos que no se tendrán en cuenta.

En resumen, dado intervalo fijado por el usuario, se calcularán intervalos cuyo comienzo estará un poco atrasado para evitar la pérdida de información.

Posteriormente se toma la fecha de referencia user_start_time, es necesario saber dónde se colocaría en la lista de S11 filtrada, por lo que se recorre la lista filtrada comparando el tiempo de usuario con el tiempo que contiene la lista. En caso de encontrar una fecha que sea mayor o igual que el tiempo de inicio fijado por el usuario se procede a analizar:

1. Ver si la diferencia entre la fecha actual y la siguiente es mayor que 60 segundos (delta de tiempo) o 0 segundos (mismas fechas).

2. Si se cumple la primera condición se añadirá al vector

Es necesario procesar los “huecos temporales” ya que entre un S11 y otro habrá eventos de topología que hay que procesar para generar posteriormente los Snapshot.

Para ello esta función devuelve dos vectores temporales, uno de tiempos de inicio (start_time) y otro de tiempos de fin (end_time). El objetivo de estos vectores start_time y end_time es pasarlos a la función generate_snapshot y procesar eventos de topología en ese intervalo temporal.

Una vez obtenidos todos los subintervalos se ejecutará un bucle en el cual se pasarán los intervalos a la función generate_snapshot para obtener las capturas.

El resultado final se puede observar en la siguiente figura, en ella podemos ver el concentrador y los Smart Meters que cuelgan de él. A través de este módulo es fácil identificar los niveles lógicos que se establecen dentro de la red PRIME.

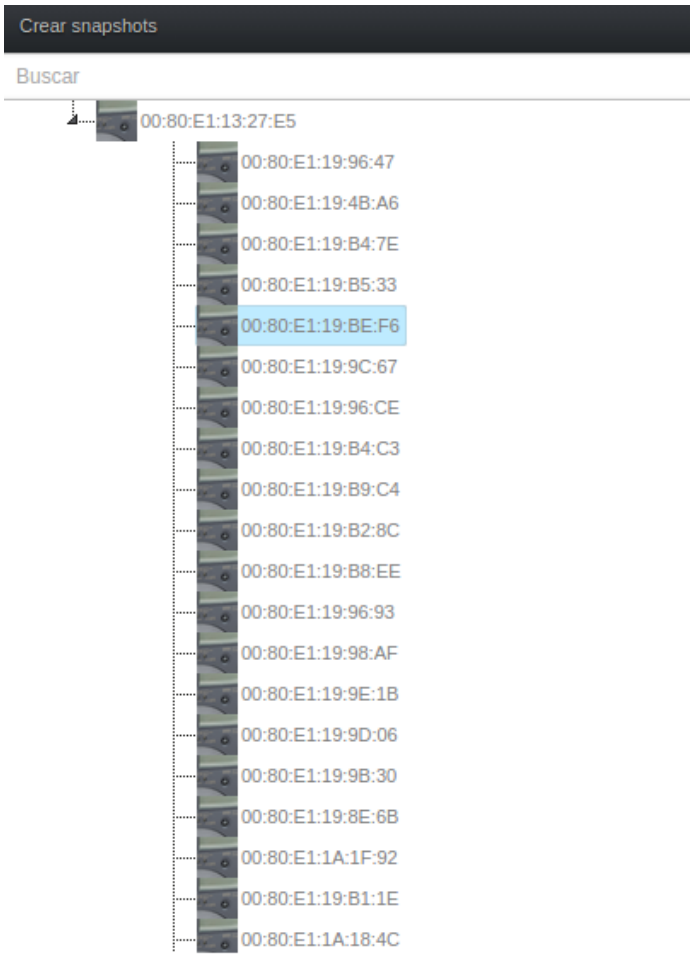



*Figura 28: Topología mostrada por el módulo desarrollado*

## 4.9 Migración a la nube

Como se expuso en capítulos anteriores, otro de los métodos de optimización consiste en utilizar lo que se conoce como computación en la nube. Básicamente evita la necesidad de utilizar recursos físicos, generalmente propios, para utilizar recursos disponibles a través de plataformas Cloud.

En este trabajo se ha elegido la plataforma ofrecida por Amazon, Amazon Web Services. El objetivo principal de utilizar esta técnica es realizar la correcta migración de la aplicación PrimeAnalytics a la nube, para conseguir optimizar el uso de recursos, el precio de los mismos y analizar cómo influye en los tiempos de respuesta del analizador.

Otro de los objetivos que se intentará conseguir, si es posible, es la implementación de un clúster en la nube con la solución distribuida presentada en los puntos anteriores. A continuación se detallarán los procedimientos seguidos para realizar la migración de la herramienta.

En primer lugar es necesario registrarse en la plataforma AWS, con la cual se obtiene la cuenta de prueba que permite hacer uso de ella durante 12 meses. En este caso se ha elegido la capa EC2, ya que es la que corresponde a la cuenta de prueba. En la siguiente figura se podrá observar la consola de administración que se ofrece para EC2, en que podemos obtener de forma rápida información sobre los recursos que estamos utilizando

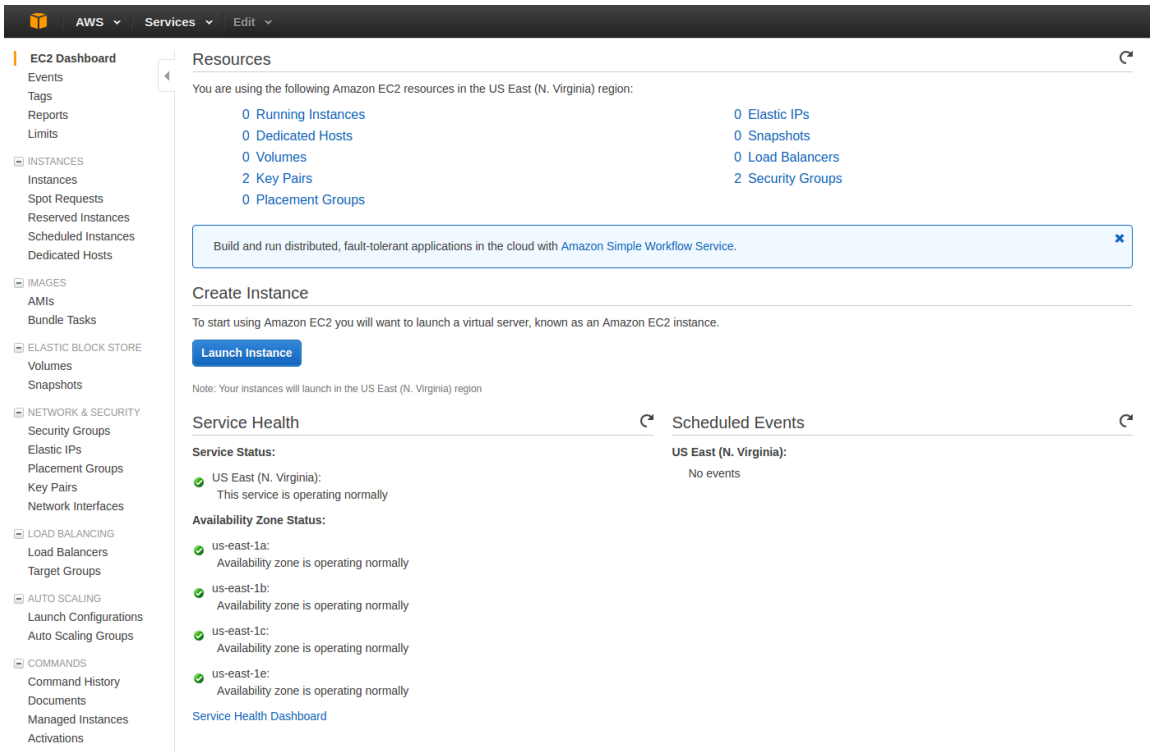



*Figura 29: Consola de administración EC2*

Tras realizar el registro y acceder a la consola de administración, el siguiente paso es lanzar la instancia EC2 en la que se aloja nuestro servidor virtual. Amazon ofrece un amplio catálogo de instancias con diferentes sistemas operativos. En la siguiente imagen se observa las más comunes. Para esta implementación se ha elegido la instancia Ubuntu Server 14.04 LTS (HVM)

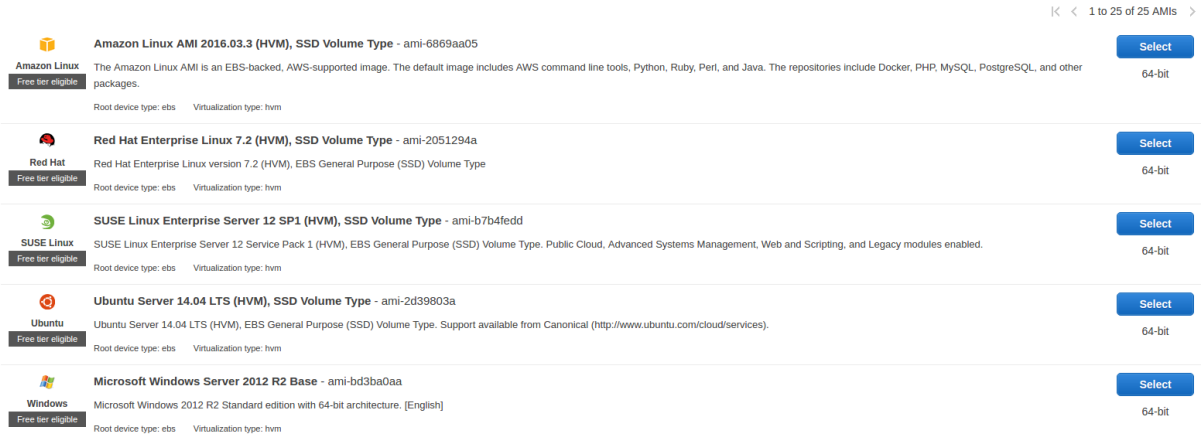



*Figura 30: Listado de S.O. disponibles*

Una vez elegida el sistema operativo en el que se basará la instancia se permite definir una serie de campos como la red de trabajo a la que pertenece, el número de instancias que se van a lanzar, el rol o la auto asignación de IPs públicas. Después de configurar este apartado se ofrece la creación de una "key pair", la cual se trata de una contraseña necesaria para la conexión vía ssh con la instancia. En la siguiente figura se observa la conexión ssh utilizando el DNS de la instancia "ec2-54.242-33-47.compute-1.amazonaws.com" y el keypair.pem definido para ella.

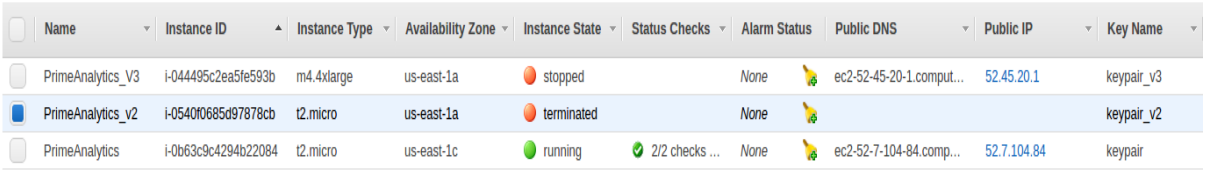
```
mario@mario-Inspiron-13-7359:~/Descargas$ chmod 400 keypair.pem
mario@mario-Inspiron-13-7359:~/Descargas$ ssh -i "keypair.pem" ubuntu@ec2-54-242-33-47.compute-1.amazonaws.com
The authenticity of host 'ec2-54-242-33-47.compute-1.amazonaws.com (54.242.33.47)' can't be established.
ECDSA key fingerprint is SHA256:jkJ2ES0Gr+Ia6hocku6dwm2+BvxrZHwxQMNryRS0Hco.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-242-33-47.compute-1.amazonaws.com,54.242.33.47' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-91-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Mon Sep 12 18:25:28 UTC 2016

  System load: 0.08              Memory usage: 5%   Processes:       82
  Usage of /:  10.0% of 7.74GB   Swap usage:   0%   Users logged in: 0

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ubuntu@ip-172-31-29-176:~$
```

*Figura 31: Conexión por SSH*

Tras realizar la conexión vía ssh con la instancia, se procede a la instalación de Docker, el docker-engine y docker-compose siguiendo los pasos proporcionados en la web oficial de Docker.

Se hace uso del programa FileZilla, el cual mediante SFTP (SSH File Transfer Protocol) transfiere a la instancia EC2 los archivos de PrimeAnalytics necesarios para su puesta en marcha. Una vez estén copiados los archivos en la instancia se puede ejecutar el comando "docker-compose up" para poner en funcionamiento el analizador de trazas.

En la siguiente figura podemos observar la información relativa al funcionamiento de la instancia en la aparece el tipo de instancia elegida, en este caso la t2.micro de la capa EC2, el estado de la misma, el DNS público así como su IP.



| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status | Public DNS | Public IP | Key Name |
|---|---|---|---|---|---|---|---|---|---|
| PrimeAnalytics_V3 | i-044495c2ea5fe593b | m4.4xlarge | us-east-1a | stopped | | None | ec2-52-45-20-1.comput... | 52.45.20.1 | keypair_v3 |
| PrimeAnalytics_v2 | i-0540f0685d97878cb | t2.micro | us-east-1a | terminated | | None | | | keypair_v2 |
| PrimeAnalytics | i-0b63c9c4294b22084 | t2.micro | us-east-1c | running | 2/2 checks ... | None | ec2-52-7-104-84.comp... | 52.7.104.84 | keypair |

*Figura 32: Estado de las instancias*

A la hora de desarrollar el analizador de trazas se definió en Django el puerto 8000 para alojarlo. Antes de acceder al él desde la instancia EC2 se han de definir una serie de reglas. Como se puede ver en la siguiente figura es necesario establecer las reglas de http, ssh, icmp y la denominada como custom tcp rule para acceder al analizador.

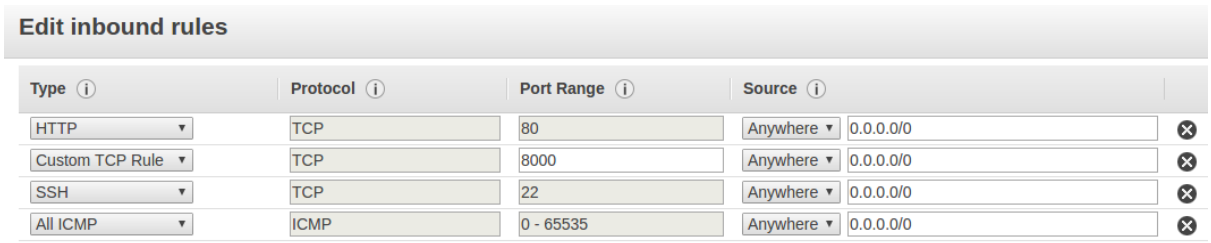



*Figura 33: reglas inbound de la instancia*

Una vez ejecutada la aplicación en la instancia elegida y utilizando el puerto 8000, se puede acceder a ella tanto desde el DNS público proporcionado en la consola de administración, o mediante la IP pública la cual se adjudica a la instancia a través del apartado denominado "Elastic IPs" perteneciente al grupo de red y seguridad de EC2.

A través de la consola también se puede obtener información de monitorización como se muestra en las siguientes figuras.

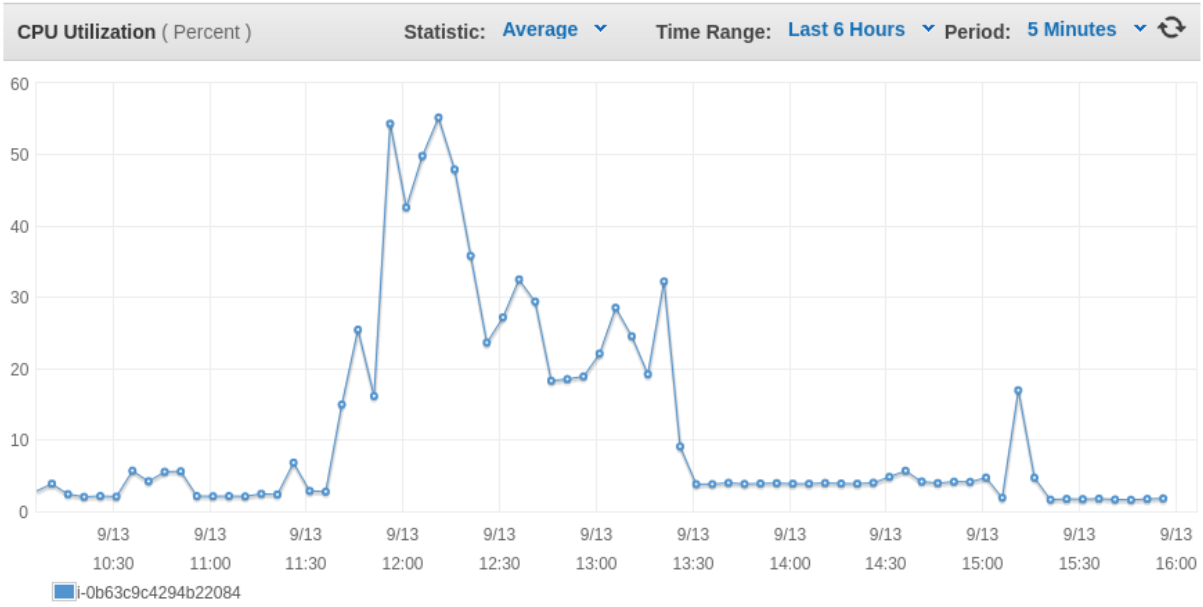



*Figura 34: Uso de la CPU*

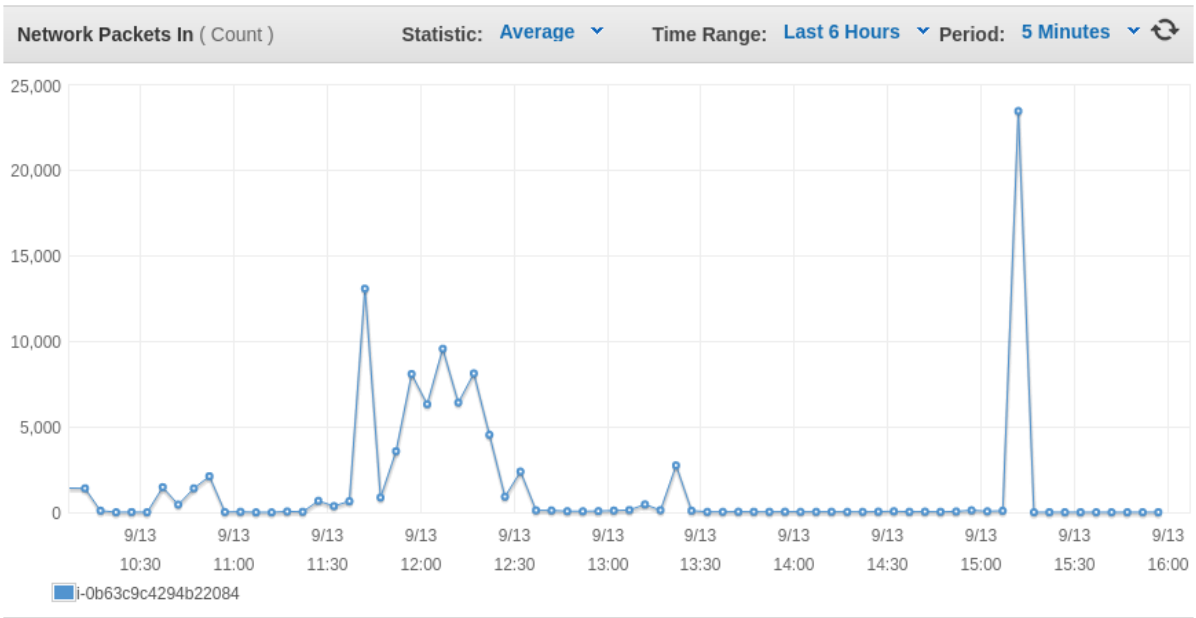



*Figura 35: Contador de paquetes entrantes*

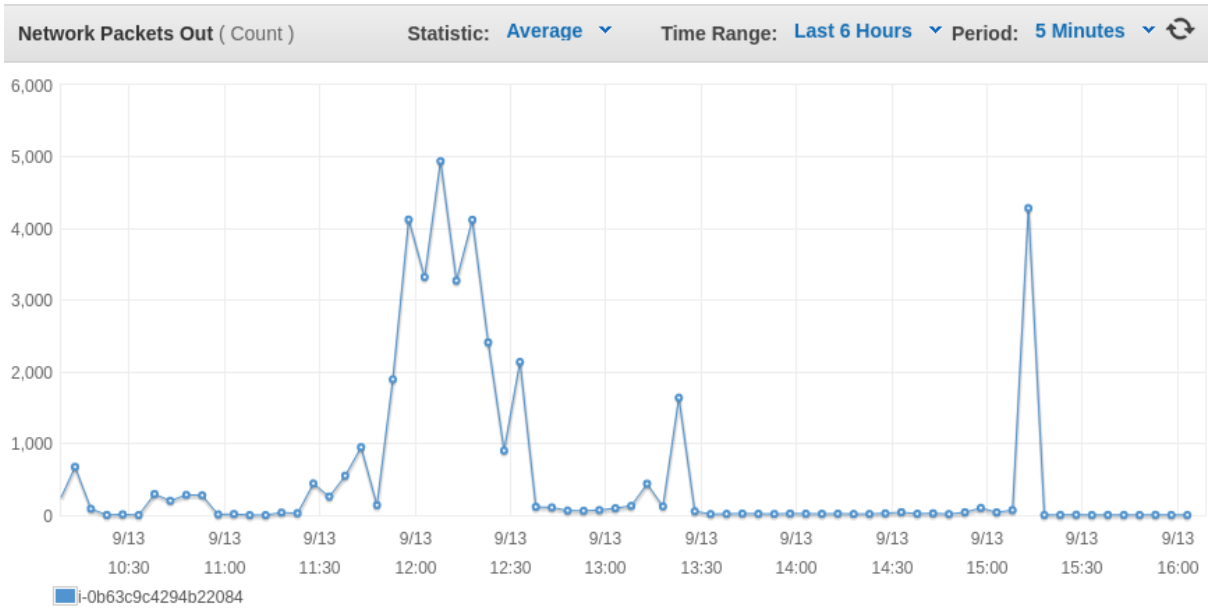



*Figura 36: Contador de paquetes salientes*

# Capítulo 5. Pruebas y evaluación

En este capítulo se muestran las pruebas realizadas para evaluar el diseño de las implementaciones, para ello se han definido dos bloques de pruebas, pruebas en entorno local y pruebas en entorno Cloud. A su vez, dentro de las pruebas en entorno local se han realizado dos métodos para obtener los resultados, el primero se conecta directamente a la base de datos para emular las consultas y el segundo mide los tiempos de respuesta de las peticiones Ajax del analizador. El objetivo es una vez obtenidos los resultados evaluar si las implementaciones consiguen reducir los tiempos de respuesta. Además de las pruebas presentadas en este capítulo se realizaron diversas pruebas que finalmente no se incluyeron debido a su extensión.

## 5.1. Entorno Local

### 5.1.1. Pruebas sobre la base de datos

Tras desarrollar e integrar con éxito la extensión Citus en la herramienta utilizando el método de distribución append, es necesario evaluar su funcionamiento para observar si existe una reducción en los tiempos de búsqueda. En este primer bloque de validación se medirá el tiempo de respuesta de la base de datos ante las consultas de la herramienta de análisis. El objetivo es evaluar si se reduce el tiempo de consulta, para ello se compara entre la base de datos sin distribuir y la base distribuida con diferentes números de Workers.

Con el fin de estandarizar y facilitar las pruebas se realizó un Script en Python, disponible en los anexos, que hacía uso de la extensión psycopg2, la cual permite conectarse a la base de datos y poder simular así las consultas que haría la aplicación.

Para esta validación se ha utilizado bases de datos distribuidas con 2, 4, 8, 16 y 40 Workers.

Se configuró la conexión a la base de datos pasando los siguientes atributos a la extensión Psycopg2:

- database
- user
- password
- host
- port

Seguidamente se declaran 4 arrays diferentes para recoger los estadísticos (media, mediana, desviación típica y varianza).

Mediante un bucle for indicamos el número de simulaciones que queremos realizar, en este caso, se han realizado 1000 simulaciones.

La simulación consiste en pasar la consulta asociada a cada gráfica directamente a la base de datos, y recogiendo el tiempo que se tarda en devolver el resultado. Una vez obtenido los tiempos de todas las simulaciones se realiza el cálculo de los estadísticos, los cuales a su vez son almacenados en los arrays declarados al inicio para posteriormente convertirlos en archivos .csv gracias a la extensión Numpy.

Una vez obtenido todos los datos en formato .csv, se utiliza la herramienta matemática Matlab para obtener las gráficas que se muestran a continuación. Para facilitar la interpretación, debajo de las tablas hay una leyenda para identificar cada número de gráfica.

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 1.07E+00 | 7.65E-01 | 6.69E-01 | 7.58E-01 | 6.72E-01 | 5.42E-01 |
| **Mediana** | 1.05E+00 | 7.11E-01 | 6.49E-01 | 6.84E-01 | 6.34E-01 | 5.34E-01 |
| **Desviación** | 7.28E-02 | 1.38E-01 | 7.11E-02 | 1.78E-01 | 1.23E-01 | 5.19E-02 |
| **Varianza** | 0.00297 | 0.00206 | 0.00237 | 0.00652 | 1.52E-02 | 2.70E-03 |

*Tabla 4: Resultados comparativos de las consultas a la BBDD en Gráfica 1*

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 3.36E-01 | 3.39E-01 | 3.32E-01 | 2.42E-01 | 2.94E-01 | 2.02E-01 |
| **Mediana** | 3.24E-01 | 3.15E-01 | 2.85E-01 | 2.34E-01 | 2.43E-01 | 1.93E-01 |
| **Desviación** | 3.12E-02 | 7.25E-02 | 1.02E-01 | 5.31E-02 | 1.06E-01 | 3.52E-02 |
| **Varianza** | 2.3e-04 | 0.00108 | 0.00076 | 0.00170 | 1.13E-02 | 1.24E-03 |

*Tabla 5: Resultados comparativos de las consultas a la BBDD en Gráfica 2*

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 5.16E-01 | 4.84E-01 | 4.68E-01 | 3.91E-01 | 4.43E-01 | 2.86E-01 |
| **Mediana** | 5.13E-01 | 4.83E-01 | 4.01E-01 | 3.47E-01 | 3.73E-01 | 2.79E-01 |
| **Desviación** | 1.47E-02 | 6.33E-02 | 1.37E-01 | 8.71E-02 | 1.46E-01 | 2.93E-02 |
| **Varianza** | 0.02529 | 0.00035 | 0.02142 | 0.00037 | 2.14E-02 | 8.60E-04 |

*Tabla 6: Resultados comparativos de las consultas a la BBDD en Gráfica 3*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 8.29E-01 | 6.59E-01 | 4.98E-01 | 5.48E-01 | 5.07E-01 | 4.29E-01 |
| **Mediana** | 8.21E-01 | 6.32E-01 | 4.85E-01 | 4.93E-01 | 4.62E-01 | 4.20E-01 |
| **Desviación** | 5.26E-02 | 5.68E-02 | 7.66E-02 | 1.19E-01 | 1.27E-01 | 4.77E-02 |
| **Varianza** | 0.02345 | 0.00014 | 0.01626 | 0.01277 | 1.63E-02 | 2.28E-03 |

*Tabla 7: Resultados comparativos de las consultas a la BBDD en Gráfica 4*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 5.45E-01 | 4.96E-01 | 3.36E-01 | 3.65E-01 | 3.32E-01 | 2.79E-01 |
| **Mediana** | 5.40E-01 | 4.94E-01 | 3.33E-01 | 3.38E-01 | 3.14E-01 | 2.74E-01 |
| **Desviación** | 4.66E-02 | 2.10E-02 | 1.41E-02 | 6.89E-02 | 1.12E-01 | 1.51E-02 |
| **Varianza** | 5.9e-04 | 0.00043 | 0.00408 | 0.00093 | 1.25E-02 | 2.30E-04 |

*Tabla 8: Resultados comparativos de las consultas a la BBDD en Gráfica 5*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 3.48E-02 | 1.55E-01 | 1.12E-01 | 1.47E-01 | 1.40E-01 | 1.08E-01 |
| **Mediana** | 3.46E-02 | 1.14E-01 | 1.08E-01 | 1.24E-01 | 1.17E-01 | 1.03E-01 |
| **Desviación** | 1.03E-03 | 6.18E-02 | 1.18E-02 | 5.00E-02 | 5.15E-02 | 1.34E-02 |
| **Varianza** | 1.55e-04 | 0.00057 | 0.00328 | 0.00081 | 2.65E-03 | 1.80E-04 |

*Tabla 9: Resultados comparativos de las consultas a la BBDD en Gráfica 6*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 3.45E-02 | 1.65E-01 | 1.20E-01 | 1.63E-01 | 2.21E-01 | 1.19E-01 |
| **Mediana** | 3.43E-02 | 1.30E-01 | 1.16E-01 | 1.37E-01 | 1.97E-01 | 1.15E-01 |
| **Desviación** | 8.00E-04 | 5.80E-02 | 1.16E-02 | 5.42E-02 | 7.42E-02 | 1.32E-02 |
| **Varianza** | 1e-04 | 0.00046 | 0.00417 | 0.00118 | 5.51E-03 | 1.70E-04 |

*Tabla 10: Resultados comparativos de las consultas a la BBDD en Gráfica 7*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 8.90E-04 | 1.31E-01 | 9.45E-02 | 1.43E-01 | 1.36E-01 | 9.51E-02 |
| **Mediana** | 8.80E-04 | 9.55E-02 | 9.05E-02 | 1.10E-01 | 1.03E-01 | 8.34E-02 |
| **Desviación** | 1.40E-04 | 5.76E-02 | 1.29E-02 | 5.85E-02 | 6.26E-02 | 3.71E-02 |
| **Varianza** | 2e-04 | 0.00063 | 0.00061 | 0.00091 | 3.92E-03 | 1.38E-03 |

*Tabla 11: Resultados comparativos de las consultas a la BBDD en Gráfica 8*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 4.34E-03 | 9.26E-02 | 6.90E-02 | 9.77E-02 | 8.87E-02 | 6.51E-02 |
| **Mediana** | 4.30E-03 | 6.34E-02 | 6.13E-02 | 7.93E-02 | 6.76E-02 | 5.73E-02 |
| **Desviación** | 4.60E-04 | 5.08E-02 | 1.93E-02 | 3.71E-02 | 3.76E-02 | 1.84E-02 |
| **Varianza** | 7e-4 | 0.00059 | 0.00066 | 0.00073 | 1.41E-03 | 3.40E-04 |

*Tabla 12: Resultados comparativos de las consultas a la BBDD en Gráfica 9*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 8.08E-01 | 6.28E-01 | 4.40E-01 | 5.39E-01 | 5.30E-01 | 3.74E-01 |
| **Mediana** | 7.67E-01 | 5.63E-01 | 4.37E-01 | 4.88E-01 | 4.80E-01 | 3.69E-01 |
| **Desviación** | 1.09E-01 | 1.56E-01 | 1.46E-02 | 1.20E-01 | 1.31E-01 | 1.77E-02 |
| **Varianza** | 0.0229 | 0.00030 | 0.02215 | 0.00498 | 1.71E-02 | 3.20E-04 |

*Tabla 13: Resultados comparativos de las consultas a la BBDD en Gráfica 10*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 8.65E-01 | 7.44E-01 | 5.35E-01 | 6.55E-01 | 6.66E-01 | 4.92E-01 |
| **Mediana** | 8.61E-01 | 6.86E-01 | 5.31E-01 | 5.82E-01 | 5.82E-01 | 4.82E-01 |
| **Desviación** | 6.06E-02 | 1.65E-01 | 1.62E-02 | 1.56E-01 | 1.97E-01 | 6.27E-02 |
| **Varianza** | 0.02815 | 0.00028 | 0.02023 | 0.0055 | 3.90E-02 | 3.94E-03 |

*Tabla 14: Resultados comparativos de las consultas a la BBDD en Gráfica 11*

A continuación se muestran una serie de gráficas en las que se pueden observar los tiempos de respuesta de la base de datos normal vs distribuida. En los anexos se incluyen las gráficas individuales de cada gráfica en función del número de Workers utilizado.

- Gráfica 1: Tipos de paquete
- Gráfica 2: Uplink vs Downlink
- Gráfica 3: Tipos de paquetes Uplink
- Gráfica 4: Tipos de paquetes Downlink
- Gráfica 5: Instantes con mayor nº de desregistros
- Gráfica 6: Switches ordenados por nº desregistros de sus hijos
- Gráfica 7: Nodos ordenados por nº de desregistros
- Gráfica 8: % de conexiones a switch
- Gráfica 9: SNR vs Tiempo
- Gráfica 10: SNR medio para cada nodo
- Gráfica 11: PNA

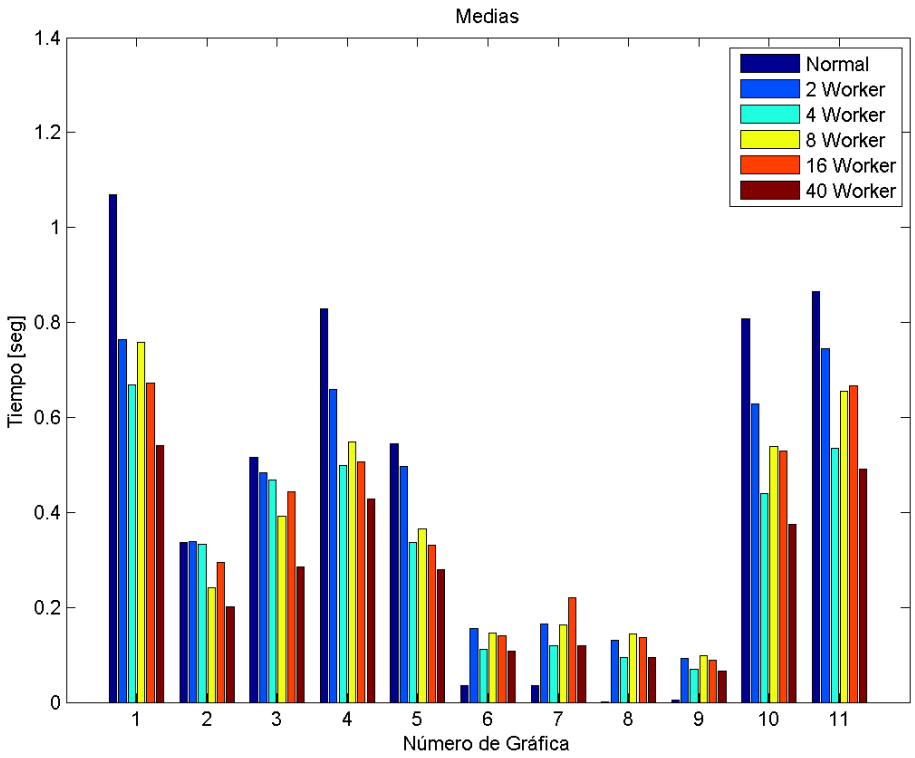



*Figura 37: Comparativa medias consulta BBDD*

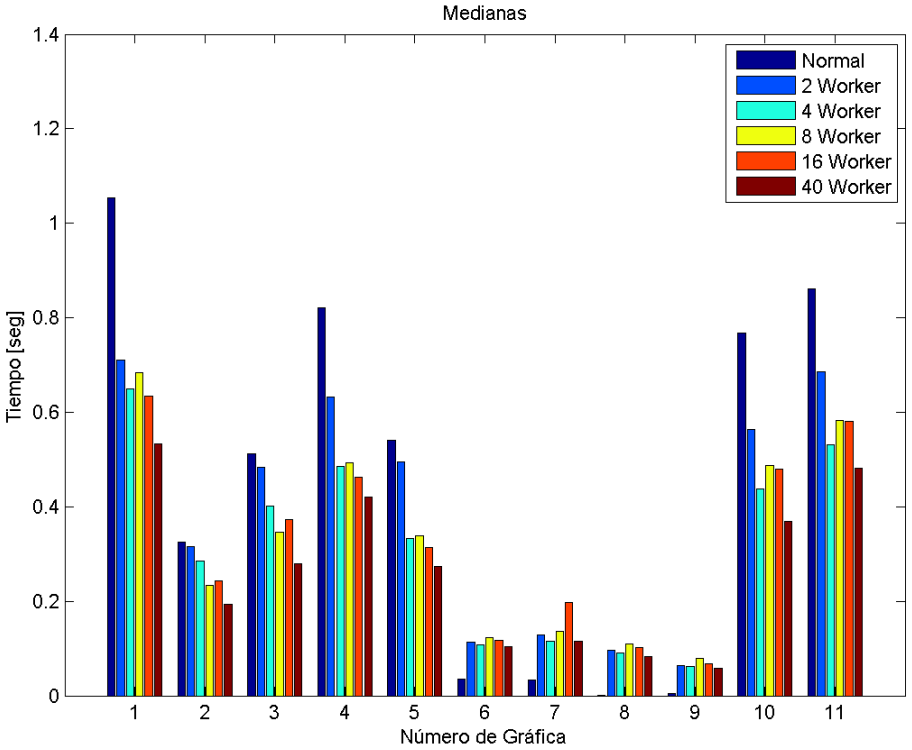



*Figura 38: Comparativa medianas consulta BBDD*

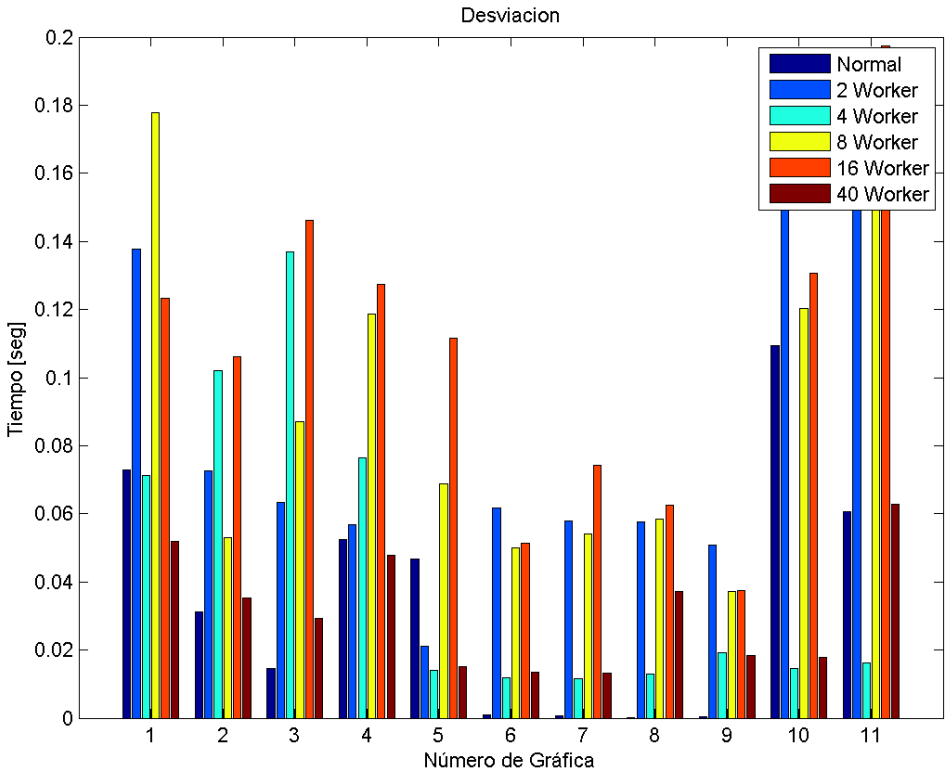



*Figura 39: Comparativa de desviación típica consulta BBDD*

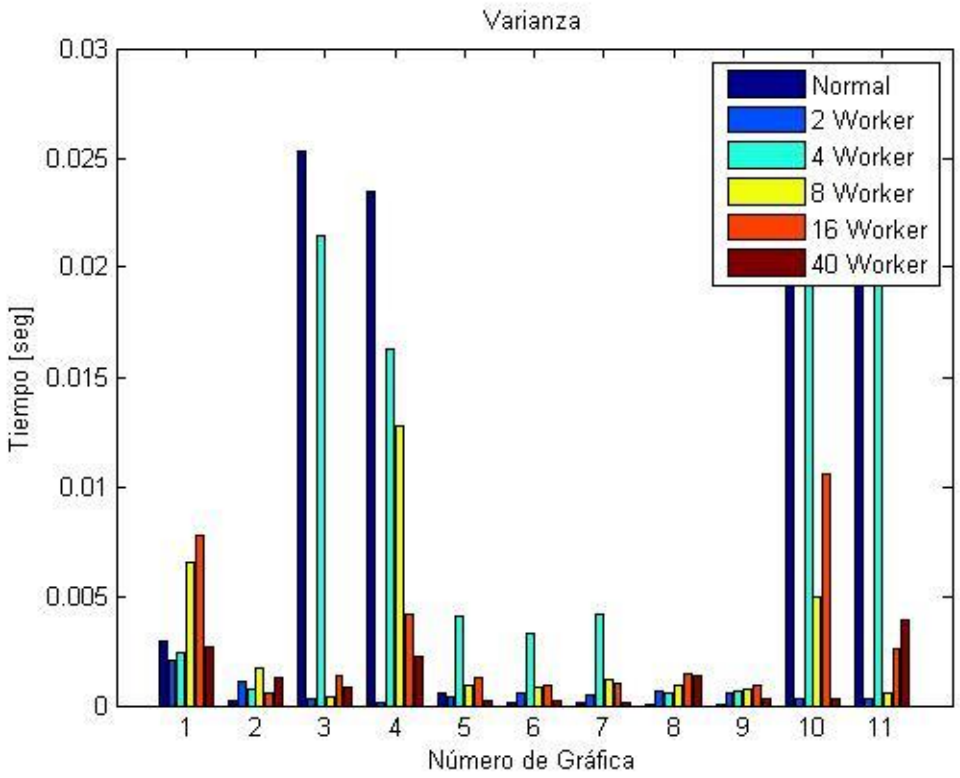



*Figura 40: Comparativa varianza consulta BBDD*

### 5.1.2. Pruebas sobre las peticiones Ajax de la herramienta

En este segundo bloque de validación, se evalúa el tiempo que tardan en generarse las gráficas en la aplicación web del analizador.

Al igual que en el primer bloque, se compara entre la base de datos sin distribuir y la base distribuida con diferentes número de Workers.

La aplicación utiliza peticiones Ajax para cargar las gráficas de manera independiente, por lo que es necesario tomar los datos relevantes de esas consultas con el fin de utilizar un script programado en Python que simula las consultas.

Para el desarrollo de dicho script se ha utilizado la extensión Http.client, la cual permite realizar una conexión http a la aplicación web del analizador de trazas. Para el correcto funcionamiento del script necesitamos saber los campos “payload” y “headers” de cada consulta Ajax, haciendo uso de la extensión de Google Chrome “Postman” [47] obtenemos estos datos.

Al igual que en el primer bloque, se declaran 4 vectores que recogen los estadísticos (media, mediana, desviación y varianza).

Mediante un bucle for indicamos el número de simulaciones que queremos realizar, en este caso, se han realizado 1000 simulaciones.

Una vez se han obtenido los tiempos de todas las simulaciones se realiza el cálculo de los estadísticos, los cuales a su vez son almacenados en los arrays declarados al inicio para posteriormente convertirlos en archivos .csv gracias a la extensión Numpy.

Finalmente obtenidos todos los datos en formato .csv, se utiliza la herramienta matemática Matlab para obtener las gráficas que se muestran a continuación. Para facilitar la interpretación, debajo de las tablas hay una leyenda para identificar cada número de gráfica

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 1.86E+00 | 1.08E+00 | 1.04E+00 | 1.02E+00 | 1.05E+00 | 5.42E-01 |
| **Mediana** | 1.85E+00 | 1.07E+00 | 1.03E+00 | 9.99E-01 | 1.03E+00 | 5.34E-01 |
| **Desviación** | 5.44E-02 | 4.54E-02 | 4.97E-02 | 8.60E-02 | 8.78E-02 | 5.19E-02 |
| **Varianza** | 2.97E-03 | 2.06E-03 | 2.22E-02 | 7.41E-03 | 7.72E-03 | 2.70E-03 |

*Tabla 15: Resultados comparativos de respuestas Ajax en Gráfica 1*

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 1.64E+00 | 1.21E+00 | 1.18E+00 | 1.12E+00 | 1.14E+00 | 2.02E-01 |
| **Mediana** | 1.64E+00 | 1.21E+00 | 1.18E+00 | 1.12E+00 | 1.14E+00 | 1.93E-01 |
| **Desviación** | 1.51E-02 | 3.29E-02 | 2.89E-02 | 4.24E-02 | 2.31E-02 | 3.52E-02 |
| **Varianza** | 2.30E-04 | 3.50E-04 | 8.40E-04 | 1.80E-03 | 5.30E-04 | 1.24E-03 |

*Tabla 16: Resultados comparativos de respuestas Ajax en Gráfica 2*

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 1.54E+00 | 8.89E-01 | 9.98E-01 | 8.30E-01 | 8.73E-01 | 2.86E-01 |
| **Mediana** | 1.48E+00 | 8.90E-01 | 9.27E-01 | 8.31E-01 | 8.65E-01 | 2.79E-01 |
| **Desviación** | 2.13E-01 | 1.88E-02 | 1.56E-01 | 2.14E-02 | 3.67E-02 | 2.93E-02 |
| **Varianza** | 4.53E-02 | 3.50E-04 | 2.44E-02 | 4.60E-04 | 1.35E-03 | 8.60E-04 |

*Tabla 17: Resultados comparativos de respuestas Ajax en Gráfica 3*

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 8.89E-01 | 6.31E-01 | 7.14E-01 | 6.44E-01 | 6.24E-01 | 4.29E-01 |
| **Mediana** | 8.27E-01 | 6.30E-01 | 6.59E-01 | 5.90E-01 | 6.06E-01 | 4.20E-01 |
| **Desviación** | 1.53E-01 | 1.19E-02 | 1.39E-01 | 1.18E-01 | 6.46E-02 | 4.77E-02 |
| **Varianza** | 2.34E-02 | 1.40E-04 | 1.93E-02 | 1.39E-02 | 4.17E-03 | 2.28E-03 |

*Tabla 18: Resultados comparativos de respuestas Ajax en Gráfica 4*

| | **Normal** | **2 Workers** | **4 Workers** | **8 Workers** | **16 Workers** | **40 Workers** |
|---|---|---|---|---|---|---|
| **Media** | 1.16E-01 | 3.02E-01 | 3.54E-01 | 3.16E-01 | 3.42E-01 | 2.79E-01 |
| **Mediana** | 1.04E-01 | 3.03E-01 | 3.38E-01 | 3.16E-01 | 3.41E-01 | 2.74E-01 |
| **Desviación** | 2.42E-02 | 2.07E-02 | 6.61E-02 | 2.70E-02 | 3.51E-02 | 1.51E-02 |
| **Varianza** | 5.90E-04 | 4.30E-04 | 4.38E-03 | 7.30E-04 | 1.24E-03 | 2.30E-04 |

*Tabla 19: Resultados comparativos de respuestas Ajax en Gráfica 5*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 9.32E-02 | 2.93E-01 | 3.60E-01 | 3.03E-01 | 3.24E-01 | 1.08E-01 |
| **Mediana** | 8.75E-02 | 2.95E-01 | 3.49E-01 | 3.04E-01 | 3.23E-01 | 1.03E-01 |
| **Desviación** | 1.23E-02 | 2.38E-02 | 6.14E-02 | 2.88E-02 | 3.05E-02 | 1.34E-02 |
| **Varianza** | 1.50E-04 | 5.70E-04 | 3.78E-03 | 8.30E-04 | 9.30E-04 | 1.80E-04 |

*Tabla 20: Resultados comparativos de respuestas Ajax en Gráfica 6*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 9.01E-02 | 3.10E-01 | 3.53E-01 | 3.24E-01 | 3.46E-01 | 1.19E-01 |
| **Mediana** | 8.80E-02 | 3.14E-01 | 3.28E-01 | 3.21E-01 | 3.45E-01 | 1.15E-01 |
| **Desviación** | 9.93E-03 | 2.14E-02 | 6.97E-02 | 3.50E-02 | 3.19E-02 | 1.32E-02 |
| **Varianza** | 1.00E-04 | 4.60E-04 | 4.87E-03 | 1.23E-03 | 1.02E-03 | 1.70E-04 |

*Tabla 21: Resultados comparativos de respuestas Ajax en Gráfica 7*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 1.37E-02 | 2.64E-01 | 2.72E-01 | 2.71E-01 | 2.87E-01 | 9.51E-02 |
| **Mediana** | 1.29E-02 | 2.61E-01 | 2.72E-01 | 2.70E-01 | 2.87E-01 | 8.34E-02 |
| **Desviación** | 4.23E-03 | 2.51E-02 | 2.85E-02 | 3.05E-02 | 3.77E-02 | 3.71E-02 |
| **Varianza** | 2.00E-05 | 6.30E-04 | 8.10E-04 | 9.30E-04 | 1.42E-03 | 1.38E-03 |

*Tabla 22: Resultados comparativos de respuestas Ajax en Gráfica 8*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 3.23E-02 | 2.79E-01 | 2.73E-01 | 2.85E-01 | 2.93E-01 | 6.51E-02 |
| **Mediana** | 2.96E-02 | 2.81E-01 | 2.76E-01 | 2.88E-01 | 2.98E-01 | 5.73E-02 |
| **Desviación** | 8.61E-03 | 2.43E-02 | 2.72E-02 | 2.79E-02 | 3.07E-02 | 1.84E-02 |
| **Varianza** | 7.00E-05 | 5.90E-04 | 7.40E-04 | 7.80E-04 | 9.40E-04 | 3.40E-04 |

*Tabla 23: Resultados comparativos de respuestas Ajax en Gráfica 9*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 1.64E+00 | 9.64E-01 | 1.27E+00 | 9.26E-01 | 9.73E-01 | 3.74E-01 |
| **Mediana** | 1.57E+00 | 9.64E-01 | 1.12E+00 | 9.06E-01 | 9.42E-01 | 3.69E-01 |
| **Desviación** | 1.82E-01 | 1.73E-02 | 3.19E-01 | 7.63E-02 | 1.03E-01 | 1.77E-02 |
| **Varianza** | 3.30E-02 | 3.00E-04 | 1.02E-01 | 5.83E-03 | 1.06E-02 | 3.20E-04 |

*Tabla 24: Resultados comparativos de respuestas Ajax en Gráfica 10*

| | Normal | 2 Workers | 4 Workers | 8 Workers | 16 Workers | 40 Workers |
|---|---|---|---|---|---|---|
| **Media** | 1.20E+00 | 7.74E-01 | 8.46E-01 | 7.27E-01 | 7.57E-01 | 4.92E-01 |
| **Mediana** | 1.10E+00 | 7.75E-01 | 7.93E-01 | 7.27E-01 | 7.47E-01 | 4.82E-01 |
| **Desviación** | 2.05E-01 | 1.68E-02 | 1.49E-01 | 2.33E-02 | 5.08E-02 | 6.27E-02 |
| **Varianza** | 4.21E-02 | 2.80E-04 | 2.22E-02 | 5.40E-04 | 2.58E-03 | 3.94E-03 |

*Tabla 25: Resultados comparativos de respuestas Ajax en Gráfica 10*

A continuación se muestran una serie de gráficas en las que se pueden observar los tiempos de respuesta de la base de datos normal vs distribuida. En los anexos se incluyen las gráficas individuales de cada gráfica en función del número de Workers utilizado.

- Gráfica 1: Tipos de paquete
- Gráfica 2: Uplink vs Downlink
- Gráfica 3: Tipos de paquetes Uplink
- Gráfica 4: Tipos de paquetes Downlink
- Gráfica 5: Instantes con mayor nº de desregistros
- Gráfica 6: Switches ordenados por nº desregistros de sus hijos
- Gráfica 7: Nodos ordenados por nº de desregistros
- Gráfica 8: % de conexiones a switch
- Gráfica 9: SNR vs Tiempo
- Gráfica 10: SNR medio para cada nodo
- Gráfica 11: PNA

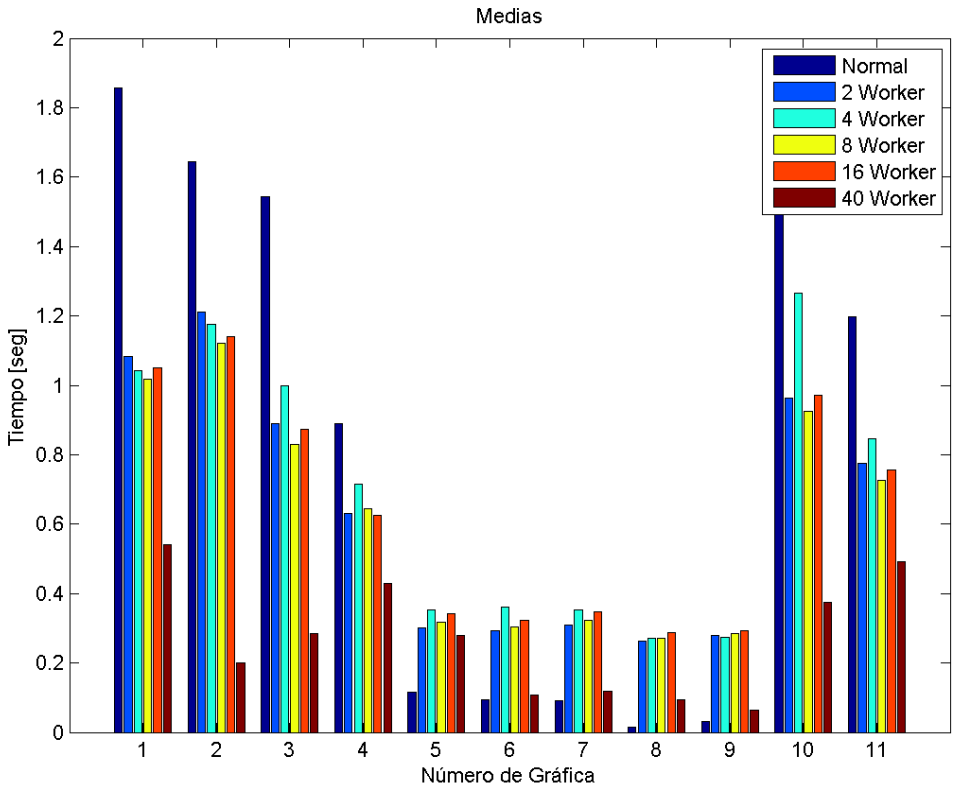



*Figura 41: Comparativa medias respuesta Ajax*

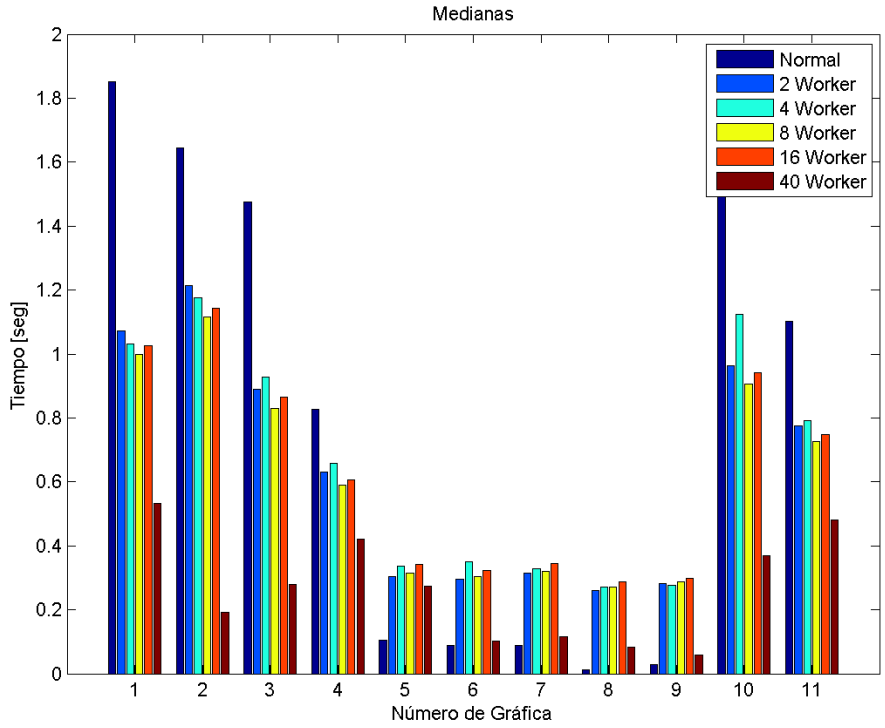



*Figura 42: Comparativa medianas respuesta Ajax*

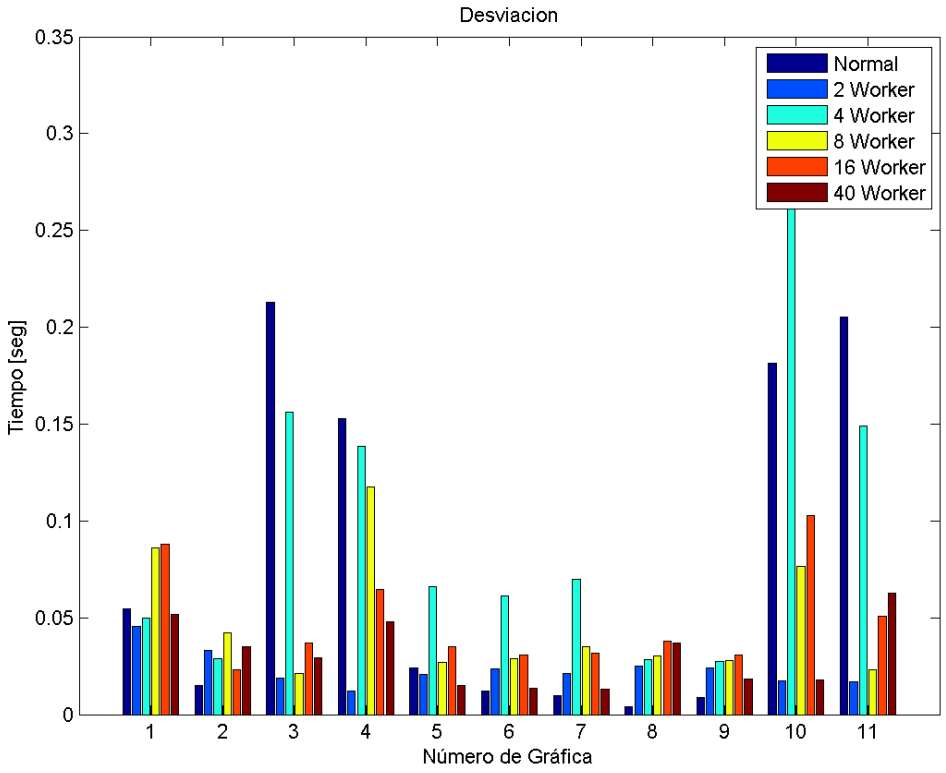



*Figura 43: Comparativa desviación típica respuesta Ajax*

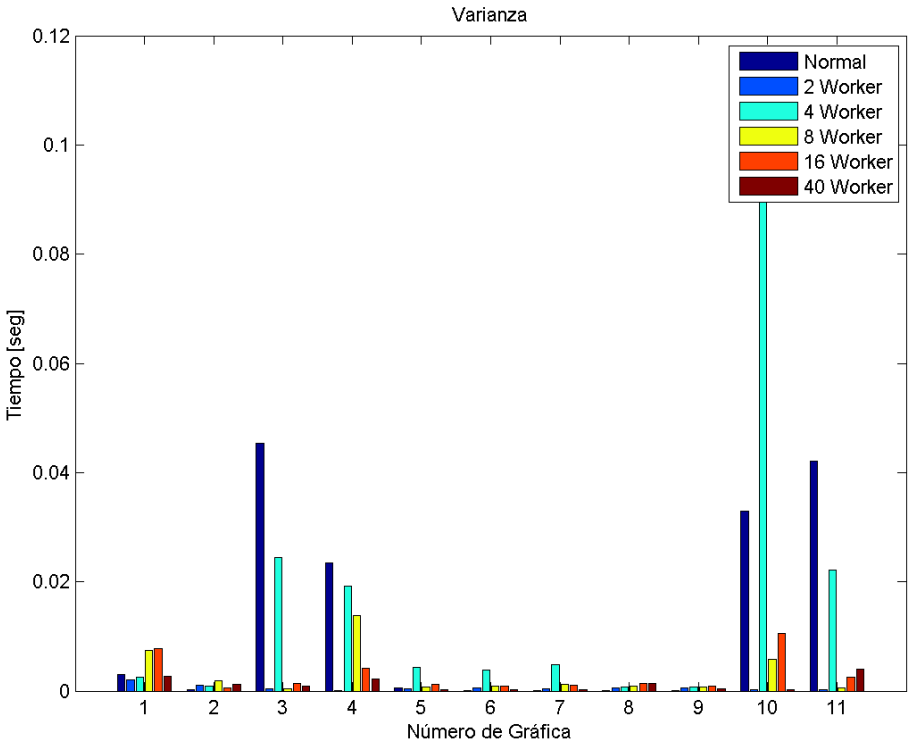



*Figura 44: Comparativa varianza respuesta Ajax*

Como puede observarse en la figura 38 y 42, los tiempos de respuesta se reducen considerablemente en las gráficas número 1, 2, 3, 4, 5, 10 y 11 al utilizar 40 Workers.

Por otro lado en las gráficas número 6, 7, 8 y 9 no se aprecia mejora en ninguno de los casos en los que se utiliza la base de datos distribuida, esto se debe principalmente a que esas gráficas hacen uso de una consulta que pide el recuento de un determinado dato en cada una de ellas. Generalmente este tipo de consultas son muy rápidas debido a que no es necesario la realización de operaciones matemáticas ni filtrados especiales por parte de la base de datos. En el caso de las BBDD distribuidas para las gráficas 6, 7, 8 y 9, no hay mejora ya que el Master debe lanzar la consulta a cada uno de los Workers y recibir la respuesta, y pese a que la obtención del resultado por parte de los Workers es rápida, si se suman los tiempos de envío de consultas y recepción de datos siempre es superior al tiempo que tarda la BBDD no distribuida en devolver este tipo de resultados.

Pese a obtener mejoras solo en 7 de las 11 gráficas analizadas, el aumento de tiempo en las 4 restantes no es muy significativo observando la reducción que se obtiene en las demás, por lo que la paralelización ha cumplido los objetivos fijados en el primer capítulo.

Cabe destacar que aparte de las pruebas y resultados mostrados en este apartado, también se obtuvo gráficas individuales de cada consulta para 100 y 1000 simulaciones, así como otro bloque idéntico de pruebas en el cual se acotaban las consultas en un rango temporal, finalmente no se incluyeron estos gráficos y resultados por ser demasiado extensos.

## 5.2. Entorno Cloud

### 5.1.2. Pruebas sobre las peticiones Ajax de la herramienta

En este apartado se han realizado pruebas similares al punto 5.1.2, en este caso se está evaluando en rendimiento de la aplicación migrada a la nube. Para ello se comparan los tiempos de respuesta Ajax de cada una de las gráficas utilizando las siguientes máquinas

Dell Inspiron 7000, equipo de laboratorio.

t2.micro (AWS), instancia gratuita.

m4.4xlarge (AWS), instancia de pago.

Características de las máquinas:

| Máquina | Procesador | RAM |
|---|---|---|
| Dell Inspiron 7000 series | Intel i7 | 8 GB |
| Instancia t2.micro | Intel Xeon | 0.5 GB |
| Instancia m4.4xlarge | Intel Xeon E5-2676 | 64 GB |

Con el fin de evaluar el rendimiento de los recursos Cloud vs los recursos físicos como pudiera ser un servidor o un ordenador portátil. A continuación se muestran una serie de gráficas que exponen los resultados obtenidos para 1000 simulaciones.

Para facilitar la interpretación, se muestra una leyenda para la numeración de las gráficas:

- Gráfica 1: Tipos de paquete
- Gráfica 2: Uplink vs Downlink
- Gráfica 3: Tipos de paquetes Uplink
- Gráfica 4: Tipos de paquetes Downlink
- Gráfica 5: Instantes con mayor nº de desregistros
- Gráfica 6: Switches ordenados por nº desregistros de sus hijos
- Gráfica 7: Nodos ordenados por nº de desregistros
- Gráfica 8: % de conexiones a switch
- Gráfica 9: SNR vs Tiempo
- Gráfica 10: SNR medio para cada nodo
- Gráfica 11: PNA

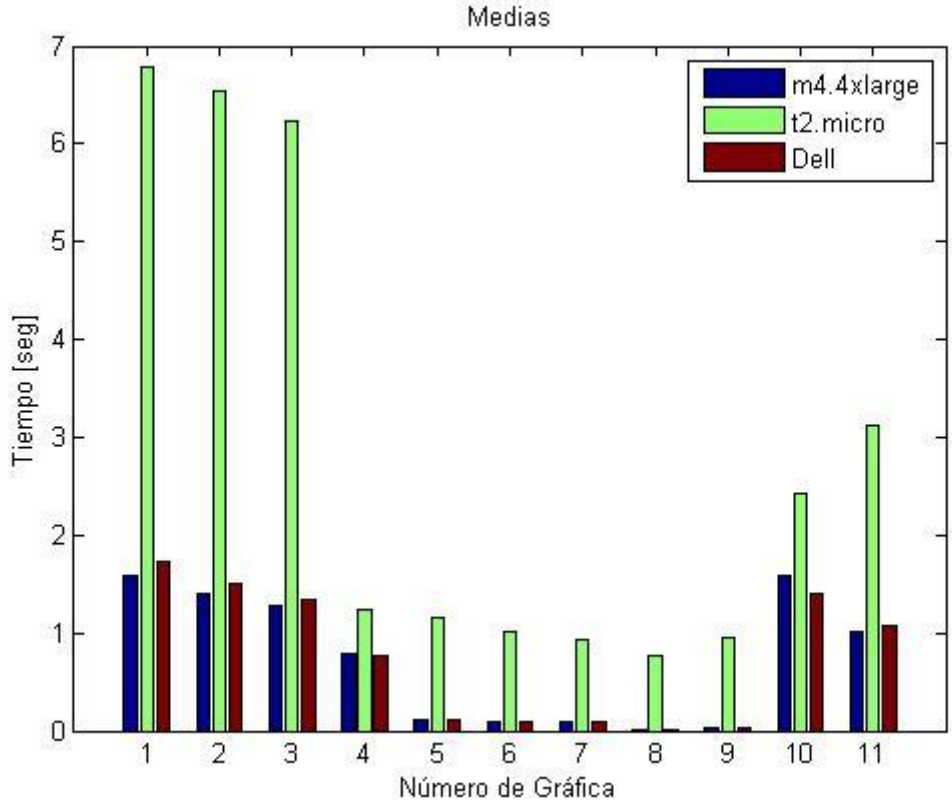



*Figura 45: Comparativa medias respuesta Ajax en Cloud*

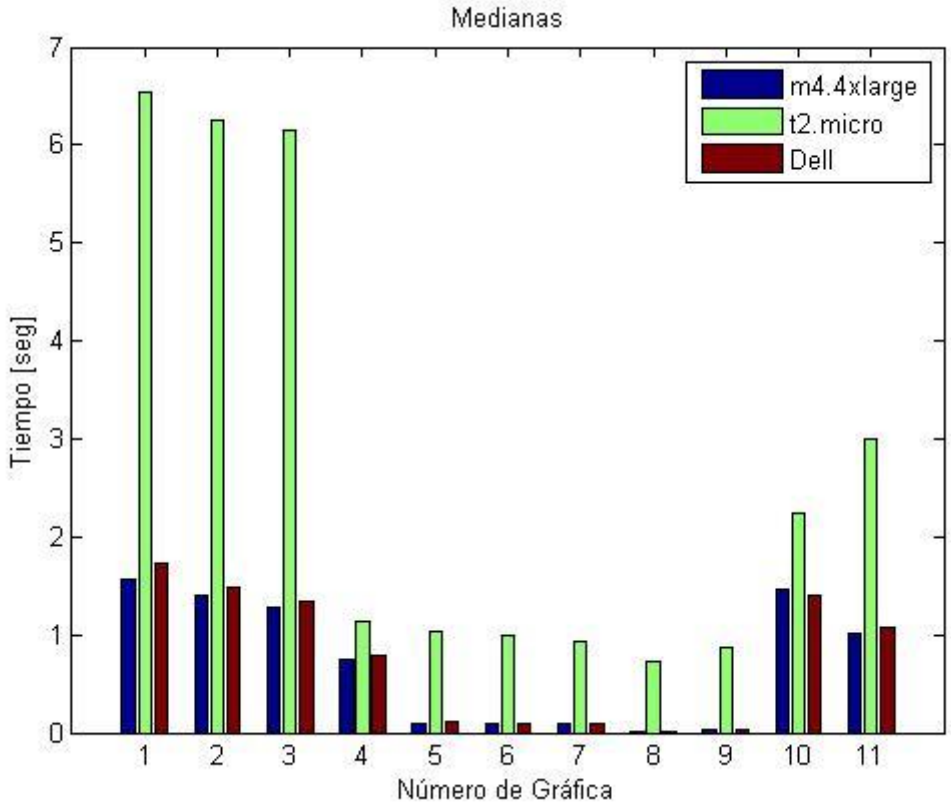



*Figura 46: Comparativa medianas respuesta Ajax en Cloud*

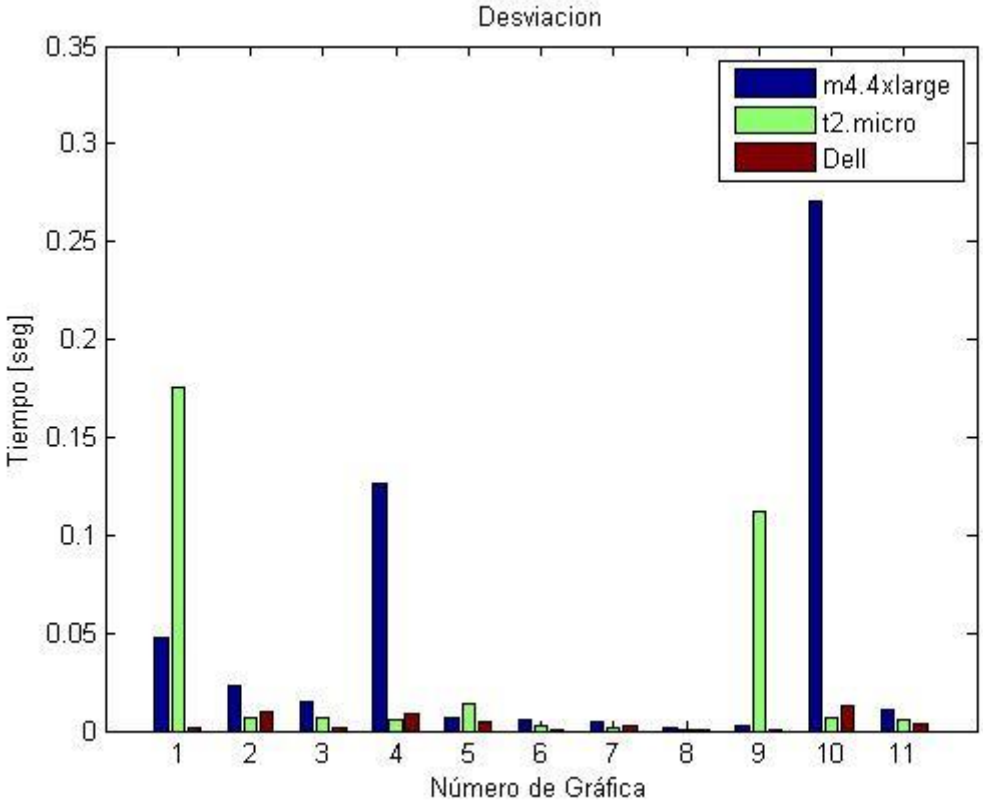



*Figura 47: Comparativa desviación respuesta Ajax en Cloud*

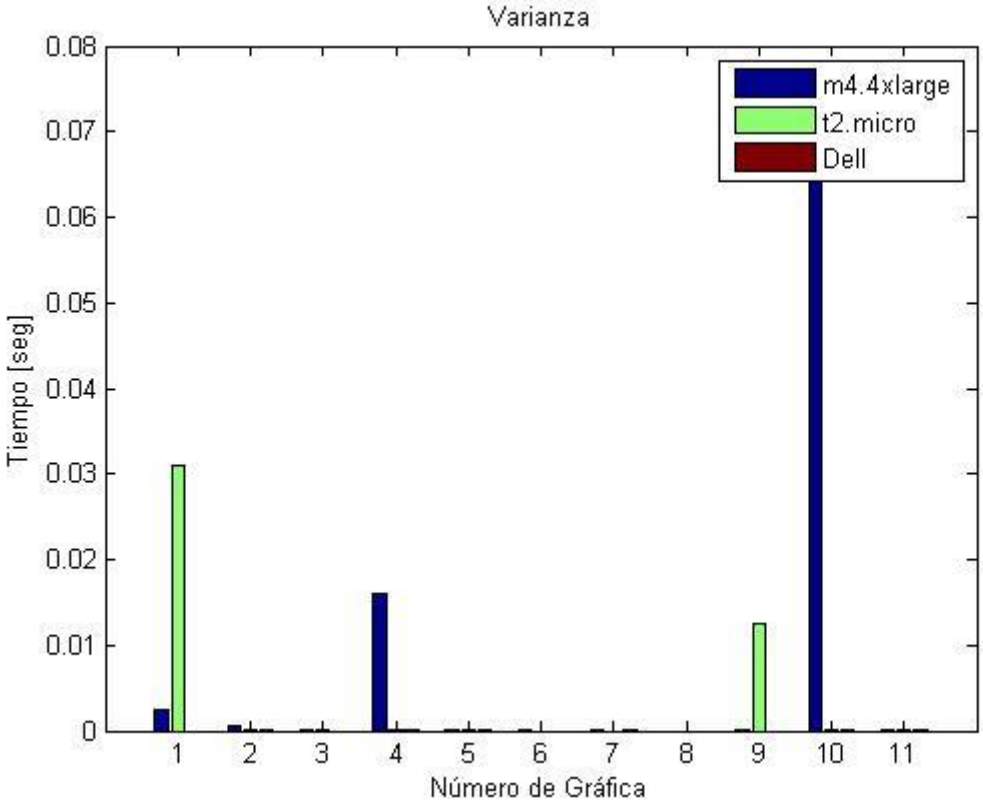



*Figura 48: Comparativa varianza respuesta Ajax en Cloud*

En este bloque de pruebas, tras analizar los resultados obtenidos se puede observar que el uso en sí de recursos Cloud no proporciona una mejora en el rendimiento, pero si se puede concluir que se puede obtener un gran rendimiento con una inversión menor, en este caso la inversión en el equipo Dell era de 700€, y con una instancia que cuesta 0,14$/hora se obtienen resultados ligeramente mejores. En la siguiente figura se puede ver el gasto realizado para las pruebas.

| Detalles | | Total |
|---|---|---|
| Cargos por servicio de AWS | | $6.32 |
| ▸ Data Transfer | | $0.00 |
| ▾ Elastic Compute Cloud | | $5.23 |
| US East (Northern Virginia) Region | Uso | |
| Amazon CloudWatch | | |
| $0.00 per alarm-month - first 10 alarms | 0.000347 Alarms | $0.00 |
| $0.00 per metric-month - first 10 metrics | 0.010 Metrics | $0.00 |
| Total: | | $0.00 |
| Amazon Elastic Compute Cloud running Linux/UNIX | | |
| $0.00 per Linux t1.micro instance-hour (or partial hour) under monthly free tier | 1 Hrs | $0.00 |
| $0.00 per Linux t2.micro instance-hour (or partial hour) under monthly free tier | 19 Hrs | $0.00 |
| $0.958 per On Demand Linux m4.4xlarge Instance Hour | 4 Hrs | $3.83 |
| Total: | | $3.83 |
| EBS | | |
| $0.00 per GB-month of General Purpose (SSD) provisioned storage under monthly free tier | 6.503 GB-Mo | $0.00 |
| $0.000 for 2000 Mbps per m4.4xlarge instance-hour (or partial hour) | 4 Hrs | $0.00 |
| Total: | | $0.00 |
| Elastic IP Addresses | | |
| $0.00 per Elastic IP address not attached to a running instance for the first hour | 1 Hrs | $0.00 |
| $0.00 per Elastic IP address remap - first 100 remaps / month | 3 Count | $0.00 |
| $0.005 per Elastic IP address not attached to a running instance per hour (prorated) | 280.800 Hrs | $1.40 |
| Total: | | $1.40 |
| Elastic Load Balancing - Classic | | |
| $0.00 per GB Data Processed by the LoadBalancer under monthly free tier | 0.000006 GB | $0.00 |
| $0.00 per LoadBalancer-hour (or partial hour) under monthly free tier | 1 Hrs | $0.00 |
| Total: | | $0.00 |
| Total de la región: | | $5.23 |

*Figura 49: Costes utilización instancias EC2*

# Capítulo 6. Conclusions and Future Works

This chapter summarizes the main conclusions after the execution of this TFG, the personal opinion and a number of lines of futures works that can begin.

## 6.1. Conclusions

In this TFG a research focused in optimizing the PrimeAnalytics tool used in analysis of PRIME networks has been conducted. It has been developed in the national research project OSIRIS (Optimization of Intelligent Monitoring Distribution Network) by the research group of the Department of Telematics UC3M. Thus, two optimization techniques were presented: parallelization of databases and cloud computing addressed.

Given that PrimeAnalytics had the aim to optimize, it was necessary to obtain a number of previous knowledge such as the Python programming language, the system Docker containers, the Django framework or system PostgreSQL databases, since the tool was developed under these characteristics.

These technical and functional knowledge were especially important to carry out both, the correct interpretation of requirements as project development, based on the achievement of the objectives set initially.

From the point of view of design and implementation of distributed solution using the extension of PostgresSQL, Citus, it was necessary to know the operation of the tool "PrimeAnalytics" and its development. After tests and evaluation of the results obtained with the simulations. It can be concluded that the use of databases distributed in the application improves the performance, since in 7 of the 11 evaluated graphs get to improve response times considerably and although in the rest of graphics, getting not reduce the time, the overall performance of the application if it improved with its use.

Thanks to the correct operation of the distributed solution, it could develop a software module, which initially had a problem because of the amount of data generated, and the

parallelization did development possible, providing a vision of the logical topology of PRIME networks. This module adds value to facilitate troubleshooting since the physical topology of these networks is a communication bus.

After getting with success the migration of PrimeAnalytics to the cloud of AWS (using for this the EC2 layer of said platform) and analyse the tests performed in this environment. It can be concluded that the use of such infrastructure is a very important thing to keep in mind when developing applications with internet access point. As seen in the graphs, with low investment can achieve results that exceed those offered by more expensive physical equipment. Notably not reached deploy the cluster in the cloud due to lack of time, however, it is proposed as a future line of work.

This TFG has covered the analysis, design, development, testing phase and evaluation of results. The design of the entire project was performed stepwise for ensure the proper development of project and avoid stagnation. Using this design was crucial as it greatly facilitated the project's progress and resolution of problems that arose during the development.

As a personal opinion I would highlight three important points after the completion of the TFG:

First, it is the great challenge posed by this TFG, long before even starting its development. This is due to important research about the different cloud platforms, tools parallelization and mainly compatibility requirements that demanded the tool. Latter, it was the most important since I could not find any example in which, an application with these characteristics, used a distributed database.

Second, is the curve of learning required to move to the design and development phase, as it was necessary to learn languages (Python, PostgreSQL) and technologies on which I didn't have experience.

I would like to give an opinion on the technical solutions used in this project. Regarding the parallelization, is true that an improvement in performance was achieved, but it was necessary to solve many problems of integration, configuration and especially choosing the correct attributes, after an investigation in documentation, forums, webinars, etc. In my opinion, the best way of getting the most out of distribution technology is to use it in applications which have a single table, typically records, with a very high volume of data used. The main problem I faced in this work was that "PrimeAnalytics" uses two different tables for the same application.

To end up with, based on tests performed and documentation consulted on cloud platforms, i think we are facing an element that should be taken into consideration when

designing and uploading applications to the Internet as an infrastructure they provide for management a company. Mainly due to the advantages in security, ease of use, monitoring or scalability as well as the price per use, which facilitates access to resources that could previously only be achieved through large investments

## 6.2. Future Works

Following the completion of this project and knowledge gained with him, I have identified two possible lines of development for the future.

The first proposal is based on using distributed databases to deploy a cluster in the cloud, using separate instances to house each of the Workers, so that they can choose the resources correctly. The great advantage of this option is the dynamic resource management as it greatly facilitates the scalability of them depending on demand. It can deploy a larger number of Workers in times when demand increases, or cut down the number of Workers in periods of inactivity, thus minimizing the expense and optimize resource usage. We are facing a line of work that could offer added value to the tool trace analysis.

The second proposal is motivated by the use of Docker containers in PrimeAnalytics, therefore, it could use the EC2 Container Service that facilitates the development and application scalability because it is a service of administration compatible with Docker. With it is possible to reduce manual administration on clusters thanks to its API. It is fully compatible with the rest of services of AWS. This line of work is a very interesting option to evolve PrimeAnalytics to a Cloud structure where distributed databases are used for operation.

# Capítulo 7. Presupuesto

Este capítulo presenta la planificación y fases de desarrollo del Trabajo de Fin de Grado así como el presupuesto del mismo, desglosado en los correspondientes costes de material y costes de recursos humanos.

## 7.1. Fases del proyecto

**Fase I: Análisis del Estado del Arte**

Durante esta fase se realizó una investigación sobre las tecnologías que habría que utilizar para la realización del proyecto, así como la familiarización con Docker, Django y Python. Para conseguir la base necesaria para el comienzo de este proyecto se necesitó el esfuerzo de un ingeniero junior durante 56 días.

**Fase II: Diseño**

Durante esta fase se estudiaron los requisitos de diseño que se debían cumplir y tomar decisiones en base a ellos. Esta fase necesitó el esfuerzo de un ingeniero junior durante 22 días.

**Fase III: Desarrollo e integración**

Durante la fase de desarrollo e integración se necesitó el esfuerzo de un ingeniero junior durante 96 días para desarrollar la mejor solución y afrontar los problemas que fueron surgiendo.

**Fase IV: Validación**

Durante esta fase, una vez integrado la solución se necesitó el esfuerzo de un ingeniero junior durante 22 días para obtener los datos de las simulaciones preparadas para obtener los datos de validación.

**Fase V: Documentación**

En esta fase fueron necesarios 26 días para documentar la totalidad de este TFG.

## 7.2. Diagrama de Gantt

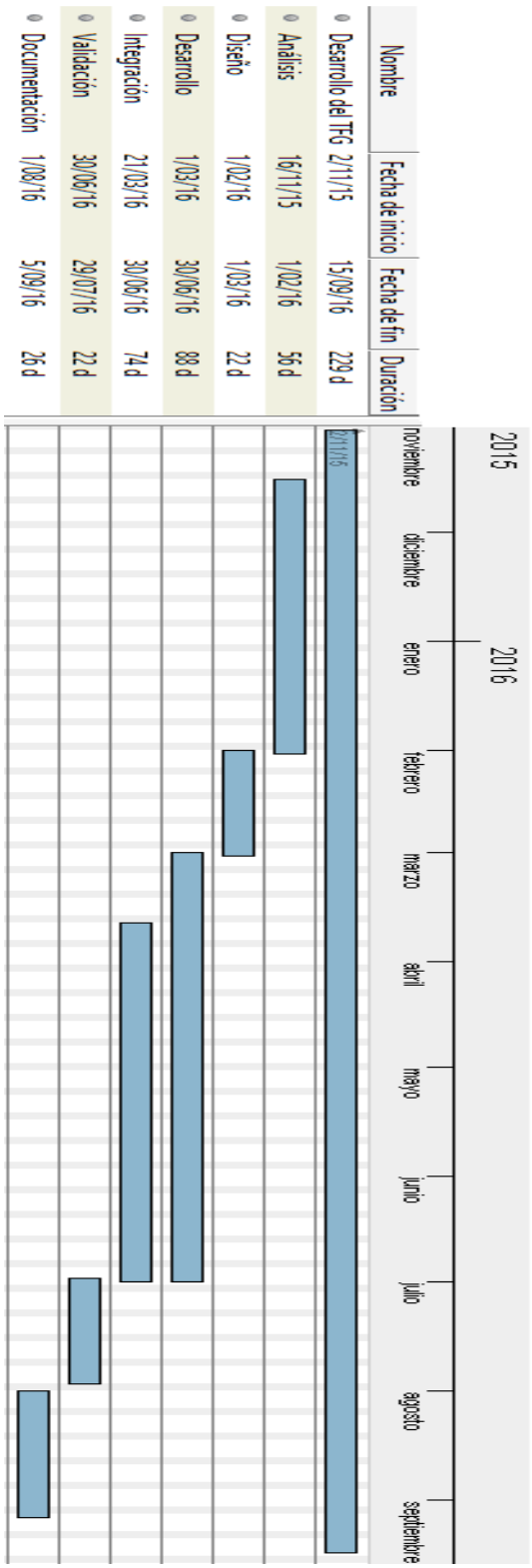



*Figura 50: Diagrama de Gantt*

## 7.1. Presupuesto

Para el desarrollo de este proyecto se ha utilizado equipamiento proporcionado por el departamento de telemática.

- Portátil Dell Inspiron 13 7000 Series
- Capa EC2 Amazon Web Services

Todo el Software utilizado para el desarrollo ha sido gratuito, ya que en las aplicaciones de pago se utilizaron las versiones de evaluación.

Para la realización de este TFG se ha utilizado un ordenador portátil Dell Inspiron 13 7000 Series. Se considera que este tipo de equipos tienen una vida útil de 48 meses, en la siguiente tabla se muestra el cálculo de amortización para el periodo durante el cual se ha utilizado este equipo, 10 meses.

| Concepto | Unidades | Coste/Unidad | Amortización (%) | Total |
|---|---|---|---|---|
| **Dell Inspiron 13** | 1 | 710€ | 15 | 106,5 € |
| **AWS** | 1 | ----- | ---- | 5,6 € |
| **Total** | | | | 112,1 € |

*Tabla 26: Costes materiales*

El cálculo del coste en recursos humanos se basa en las horas de trabajo, que son calculadas considerando el esfuerzo de un ingeniero para llevar a cabo todas las fases del proyecto descritas en el punto 6.2 de esta sección. La duración total de este TFG es de 229 días, asumiendo una jornada de 8 horas por día se obtiene el total de 1.832 horas. En la siguiente tabla se observan los resultados del cálculo.

| Categoría | Horas trabajadas | Coste/Hora | Total |
|---|---|---|---|
| **Ingeniero Junior** | 1832 | 20€/hora | |
| **Total** | | | 36.640 € |

*Tabla 27: Costes recursos humanos*

Al resumen total de gastos se ha aplicado un impuesto de costes indirectos del 20% reflejado en la tabla 28.

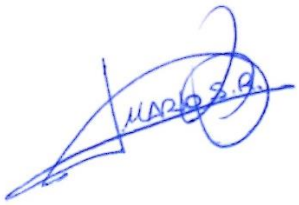
| Concepto | Total |
|---|---|
| **Costes materiales** | 112,1 € |
| **Costes recursos humanos** | 36.640 € |
| **Total** | 36.752,1 € |
| **Total (+20%)** | 44.192,10 € |

*Tabla 28: Costes totales*

Luego:

"El presupuesto total del proyecto asciende a la cantidad de **CUARENTA Y CUATRO MIL CIENTO NOVENTA Y DOS CON DIEZ** euros".

Leganés, Octubre de 2016



Mario Sanz Rodrigo

## 8. Referencias


[1] International Energy Agency, “Technology Roadmaps: Smart Grids”, 2011.

[2] G. López López, “Contribution to Machine-to-Machine Architectures for Smart Grids”. Tesis Doctoral. Universidad Carlos III de Madrid, 2014.

[3] G. López, J. I. Moreno, H. Amaris, F. Salazar, “Paving the Road toward Smart Grid through Large-Scale Advanced Metering Infrastructures”, Electric Power Systems Research, 2014. DOI: 10.1016/j.epsr.2014.05.006.

[4] «Página web oficial de la PRIME Alliance» [En línea]. Disponible: http://www.prime-alliance.org/
[Último acceso]: 24.06.2016

[5] ITU-T Standard G.9904, “Narrowband orthogonal frecuency division multiplexing power line comunication transceivers for PRIME networks”, 2012.

[6] M. Seijo, G. López, J.I. Moreno, J. Matanza, F. Martín, C. Martínez, "Herramientas TIC para la mejora del rendimiento y la planificación de redes PLC para Smart Grids", XII Jornadas de Ingeniería Telemática - JITEL 2015. Palma de Mallorca, Spain, 2015.

[7] «Página web oficial del proyecto OSIRIS» [En línea]. Disponible: http://www.unionfenosadistribucion.com/es/redes+inteligentes/investigacion+y+desarrollo/nacionales/1297262412251/osiris.html
[Último acceso]: 24.08.2016

[8] G. López, P. Moura, V. Custodio, J. I. Moreno, “Modeling the Neighborhood Area Networks of the Smart Grid”, IEEE ICC 2012, Ottawa, Canada, 2012.

[9] G. Deconinck, «An evaluation of two-way communication means for advanced metering in Flanders (Belgium),» de IEEE Int. Conf. on Instrumentation and Measurement Technology, Victoria, Vancouver Island, Canada, 2008.

[10] KEMA Consulting, «Smart Meter Requirements, Dutch Smart Meter specification and tender dossier.,» 02 2008.

[11] A. Ankou, G. Romero y G. Mauri, «Design of the overall system architecture, Tech. Rep. 3.1» de OPEN meter Consortium, 02 02 2010.

[12] T. Schaub, «Powerline Carrier - The basis for advanced metering,» de 19th International Conference on Electricity Distribution, Viena, 2007.

[13] «Malaga smartcity» [En línea]. Disponible: http://www.smartcitymalaga.es/
[Último acceso]: 03.12.2015

[14] A. Aidine, A. Tabone y J. Muller, «Deployment of Power Line Communication by European Utilities in Advanced Metering Infrastructure,» de IEEE ISPLC 2013, Johannesburg, South Africa, 2013.

[15] V. Oksman y J. Zhang, «G.hnem: the new itu-t standard on narrowband plc technology,» Communications Magazine, IEEE, vol. 49, no 12, pp. 36-44, 09 2011.

[16] «What is Python? Executive Summary | Python.org» [En línea]. Disponible: https://www.python.org/doc/essays/blurb/
[Último acceso]: 06.04.2016

[17] «Wikipedia Python» [En línea]. Disponible: https://es.wikipedia.org/wiki/Python
[Último acceso]: 06.09.2016
[18] «Python2orPython3 - Python Wiki» [En línea]. Disponible:
https://wiki.python.org/moin/Python2orPython3
[Último acceso]: 07.09.2016
[19] «Picking an Interpreter - The Hitchhiker's Guide to Python» [En línea]. Disponible:
http://docs.python-guide.org/en/latest/starting/which-python/
[Último acceso]: 20.04.2016
[20] «Extensión Psycopg» [En línea]. Disponible: http://initd.org/psycopg/features/ .
[Último acceso]: 02.09.2016
[21] «Extensión Statistics» [En línea]. Disponible:
https://docs.python.org/3/library/statistics.html
[Último acceso]: 02.09.2016
[22] «Extensión Numpy» [En línea]. Disponible: http://www.numpy.org/
[Último acceso]: 02.09.2016
[23] «Wikipedia Numpy» [En línea]. Disponible: https://es.wikipedia.org/wiki/NumPy .
[Último acceso]: 02.09.2016
[24] «Extensión Numpy (2)» [En línea]. Disponible: http://numpy.scipy.org/
[Último acceso]: 02.09.2016
[25] «Documentación Python» [En línea]. Disponible:
https://docs.python.org/3/library/subprocess.html
[Último acceso]: 02.09.2016
[26] «Documentación Python 3.1» [En línea]. Disponible:
https://docs.python.org/3.1/library/http.client.html
[Último acceso]: 02.09.2016
[27] «About the Django Software Foundation | Django» [En línea] Disponible:
https://www.djangoproject.com/foundation/
[Último acceso]: 09.09.2016
[28] «Django Overview | Django» [En línea]. Disponible:
https://www.djangoproject.com/start/overview/
[Último acceso]: 09.09.2016
[29] «Wikipedia PostgreSQL» [En línea]. Disponible:
https://es.wikipedia.org/wiki/PostgreSQL
[Último acceso]: 01.09.2016
[30] «Sitio Web PostgreSQL» [En línea]. Disponible: https://www.postgresql.org/about/ .
[Último acceso]: 02.09.2016
[31] «Docker Engine» [En línea]. Disponible: https://docs.docker.com/engine/quickstart/ .
[Último acceso]: 12.08.2016
[32] «Docker Compose» [En línea]. Disponible:
https://docs.docker.com/compose/overview/
[Último acceso]: 12.08.2016
[33] «Docker Machine» [En línea]. Disponibe: https://docs.docker.com/machine/overview/

[Último acceso]: 12.08.2016
[34] «Sitio Web PgPool» [En línea]. Disponible: http://www.pgpool.net/docs/latest/pgpool-en.html
[Último acceso]: 20.04.2016
[35] «Extensión pgshard» [En línea]. Disponible: http://www.databasesoup.com/2014/12/whats-this-pgshard-thing.html
[Último acceso]: 20.08.2016
[36] «Pg_shard Github» [En línea]. Disponible: https://github.com/citusdata/pg_shard .
[Último acceso]: 20.08.2016
[37] «Citus Data» [En línea]. Disponible: https://www.citusdata.com/
[Último acceso]: 10.09.2016
[38] «Citus Data» [En línea]. Disponible: https://docs.citusdata.com/en/v5.2/
[Último acceso]: 10.09.2016
[39] «FIWARE» [En línea]. Disponible: https://www.fiware.org/
[Último acceso]: 20.02.2016
[40] «IaaS Wikipedia» [En línea]. Disponible: https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube
[Último acceso]: 04.09.2016
[41] «PaaS Wikipedia» [En línea]. Disponible: https://en.wikipedia.org/wiki/Platform_as_a_service
[Último acceso]: 04.09.2016
[42] «SaaS Wikipedia» [En línea]. Disponible: https://es.wikipedia.org/wiki/Software_como_servicio
[Último acceso]: 04.09.2016
[43] «Webinar SoftLayer» [En línea]. Disponible: http://es.slideshare.net/IgnacioDaza/softlayer-webinar-12-jun2014
[Último acceso]: 23.08.2016
[44] «SoftLayer» [En línea]. Disponible: http://www.softlayer.com/es/
[Último acceso]: 20.08.2016
[45] «Google Cloud Platform» [En línea]. Disponible: https://cloud.google.com/
[Último acceso]: 01.09.2016
[46] «Amazon Web Services» [En línea] Disponible: http://aws.amazon.com
[Último acceso]: 10.09.2016
[47] «GetPostman» [En línea]. Disponible: https://www.getpostman.com/
[Último acceso]: 20.07.2016
[48] «Microsoft Azure» [En línea]. Disponible: https://azure.microsoft.com/es-es/
[Último acceso]: 22.07.2016
[49] «ORM« [En línea]. Disponible: https://tutorial.djangogirls.org/es/django_orm/
[Último acceso]: 30.07.2016

# Anexo I

## Marco Regulatorio.

Este Trabajo de Fin de Grado se ha centrado en la evaluación, desarrollo y validación de distintas técnicas de optimización para aplicarlas a la herramienta de análisis de trazas PrimeAnalytics.

Dado que los datos facilitados por Unión Fenosa para procesarlos en PrimeAnalytics pertenecen a escenarios reales, las modificaciones realizadas en la herramienta deben cumplir (y cumplen) con las disposiciones establecidas en la LOPD (Ley Orgánica de Protección de Datos). La Ley Orgánica 15/1999, del 13 de diciembre, de Protección de Datos de Carácter Personal, publicada en el BOE núm. 298, de 14/12/1999 y entrada en vigor desde el 14/01/2000 tiene por objeto garantizar y proteger, en lo que concierte al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar.

Debido a que el completo desarrollo de este proyecto es de carácter software, y que inicialmente no está destinado a su comercialización, cumple con la LOPD anteriormente citada y la política de buenas prácticas aplicada al desarrollo del código necesario para llevar a cabo los objetivos de este proyecto.

# ANEXO II

## Script I de validación

A continuación se muestras el script creado para la primera fase de la validación.

En el se muestra el uso de las extensiones Python para conectarse a la base de datos de la herramienta y simular las consultas que lanza la aplicación web a la BBDD así como el uso de diferentes librerías para calcular los estadísticos necesarios.

```
import psycopg2
import time
import statistics
import numpy
import subprocess

simulaciones = 1001

database = psycopg2.connect(database="primeanalytics",user="primeanalytics",password="1234",host="172.18.0.3",port="5432")
cursor = database.cursor()

array_media = []
array_mediana = []
array_desviacion = []
array_varianza = []

start = time.time()

print('----> Iniciando pruebas Benchmark sobre BBDD PrimeAnalytics <----')

# Tipos de paquete
tiempo1 = []
for i in range(1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT "trace_parser_app_traces"."specific_type", COUNT("trace_parser_app_traces"."specific_type") AS
"count" FROM "trace_parser_app_traces" WHERE scenario = 'prueba' GROUP BY "trace_parser_app_traces"."specific_type""')
        tiempo1.append(time.time() - start_time)
        database.commit()

mediana = round(statistics.median(tiempo1),5)
media = round(statistics.mean(tiempo1),5)
desviacion = round(numpy.std(tiempo1),5)
varianza = round(statistics.variance(tiempo1),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 1 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# UP vs DW
tiempo2 = []
for i in range (1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT count(*) FROM trace_parser_app_traces WHERE "do" = TRUE AND "do" = FALSE"')
        tiempo2.append(time.time() - start_time)
        database.commit()

mediana = round(statistics.median(tiempo2),5)
media = round(statistics.mean(tiempo2),5)
desviacion = round(numpy.std(tiempo2),5)
varianza = round(statistics.variance(tiempo2),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 2 <-----------')
```

```
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


# Tipos de paquete UP
tiempo3 = []
for i in range (1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT specific_type,count(*) FROM trace_parser_app_traces WHERE "do" = TRUE group by
specific_type"')
        tiempo3.append(time.time() - start_time)
        database.commit()

mediana = round(statistics.median(tiempo3),5)
media = round(statistics.mean(tiempo3),5)
desviacion = round(numpy.std(tiempo3),5)
varianza = round(statistics.variance(tiempo3),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 3 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

#Tipos de paquete DW
tiempo4 = []
for i in range (1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT "trace_parser_app_traces"."specific_type", COUNT("trace_parser_app_traces"."specific_type") AS
"count" FROM "trace_parser_app_traces" WHERE "do" = False GROUP BY "trace_parser_app_traces"."specific_type""')
        tiempo4.append(time.time() - start_time)
        database.commit()

mediana = round(statistics.median(tiempo4),5)
media = round(statistics.mean(tiempo4),5)
desviacion = round(numpy.std(tiempo4),5)
varianza = round(statistics.variance(tiempo4),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 4 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# Instantes con mayor numero de desregistros
tiempo5 = []
for i in range(1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT "trace_parser_app_traces"."specific_type", COUNT("trace_parser_app_traces"."specific_type") AS
"count" FROM "trace_parser_app_traces" WHERE "do" = True GROUP BY "trace_parser_app_traces"."specific_type""')
        tiempo5.append(time.time() - start_time)
        database.commit()

mediana = round(statistics.median(tiempo5),5)
```

```
media = round(statistics.mean(tiempo5),5)
desviacion = round(numpy.std(tiempo5),5)
varianza = round(statistics.variance(tiempo5),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 5 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


# Switches ordenados por numero de desregistros de sus hijos
tiempo6 = []
for i in range(1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT "trace_parser_app_node"."switch_mac", COUNT("trace_parser_app_node"."state") AS
"disconnections" FROM "trace_parser_app_node" WHERE
state=0 AND NOT file_type = 'SNAPSHOT' GROUP BY switch_mac ORDER BY "disconnections" DESC"')
        tiempo6.append(time.time() - start_time)
        database.commit()


mediana = round(statistics.median(tiempo6),5)
media = round(statistics.mean(tiempo6),5)
desviacion = round(numpy.std(tiempo6),5)
varianza = round(statistics.variance(tiempo6),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 6 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


# Nodos ordenados por numero de desregistros
tiempo7 = []
for i in range(1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT "trace_parser_app_node"."mac", COUNT("trace_parser_app_node"."state") AS "disconnections"
FROM "trace_parser_app_node" WHERE
state = 0 AND NOT file_type = 'SNAPSHOT' GROUP BY "trace_parser_app_node"."mac" ORDER BY "disconnections" DESC"')
        tiempo7.append(time.time() - start_time)
        database.commit()


mediana = round(statistics.median(tiempo7),5)
media = round(statistics.mean(tiempo7),5)
desviacion = round(numpy.std(tiempo7),5)
varianza = round(statistics.variance(tiempo7),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 7 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')
```

```
# % de conexiones a switches
tiempo8 = []
for i in range(1,simulaciones):
		cursor.execute('"SET citus.remote_task_check_interval TO 1"')
		start_time = time.time()
		cursor.execute('"SELECT "trace_parser_app_node"."switch_mac", COUNT("trace_parser_app_node"."switch_mac") AS
"number_switch_connections" FROM "trace_parser_app_node" WHERE
mac = '00:80:E1:18:65:A1' AND state = 1  AND NOT file_type = 'SNAPSHOT'
GROUP BY switch_mac ORDER BY "number_switch_connections" DESC"')
		tiempo8.append(time.time() - start_time)
		database.commit()

mediana = round(statistics.median(tiempo8),5)
media = round(statistics.mean(tiempo8),5)
desviacion = round(numpy.std(tiempo8),5)
varianza = round(statistics.variance(tiempo8),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n----------> Grafica 8 <----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# SNR vs Tiempo
tiempo9 = []
for i in range(1,simulaciones):
		cursor.execute('"SET citus.remote_task_check_interval TO 1"')
		start_time = time.time()
		cursor.execute('"SELECT "trace_parser_app_traces"."physnr", "trace_parser_app_traces"."pctime" FROM
"trace_parser_app_traces"
WHERE physnr >= 0 AND reg_mac = '00:80:E1:18:65:A1'  ORDER BY "trace_parser_app_traces"."pctime" ASC"')
		tiempo9.append(time.time() - start_time)
		database.commit()

mediana = round(statistics.median(tiempo9),5)
media = round(statistics.mean(tiempo9),5)
desviacion = round(numpy.std(tiempo9),5)
varianza = round(statistics.variance(tiempo9),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n----------> Grafica 9 <----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# PNA count
tiempo10 = []
for i in range(1,simulaciones):
		cursor.execute('"SET citus.remote_task_check_interval TO 1"')
		start_time = time.time()
		cursor.execute('"SELECT "trace_parser_app_traces"."pna", COUNT("trace_parser_app_traces"."pna") AS "count" FROM
"trace_parser_app_traces" WHERE scenario = 'prueba' AND NOT pna IS NULL GROUP BY "trace_parser_app_traces"."pna" ORDER BY
"count" DESC LIMIT 25"')
		tiempo10.append(time.time() - start_time)
		database.commit()

mediana = round(statistics.median(tiempo10),5)
```

```
media = round(statistics.mean(tiempo10),5)
desviacion = round(numpy.std(tiempo10),5)
varianza = round(statistics.variance(tiempo10),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 10 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


# SNR medio para cada nodo
tiempo11 = []
for i in range(1,simulaciones):
        cursor.execute('"SET citus.remote_task_check_interval TO 1"')
        start_time = time.time()
        cursor.execute('"SELECT "trace_parser_app_traces"."reg_mac", COUNT("trace_parser_app_traces"."physnr") AS "data_num",
AVG("trace_parser_app_traces"."physnr") AS "avg_snr"
FROM "trace_parser_app_traces" WHERE physnr >= 0 GROUP BY "trace_parser_app_traces"."reg_mac" ORDER BY "avg_snr" DESC"')
        tiempo11.append(time.time() - start_time)
        database.commit()

mediana = round(statistics.median(tiempo11),5)
media = round(statistics.mean(tiempo11),5)
desviacion = round(numpy.std(tiempo11),5)
varianza = round(statistics.variance(tiempo11),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 11 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


end = time.time() - start


print('\nTiempo total: \t', round(end,3),'\t seg')


a1 = numpy.asarray(tiempo1)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_1.csv",a1,delimiter=",")
a2 = numpy.asarray(tiempo2)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_2.csv",a2,delimiter=",")
a3 = numpy.asarray(tiempo3)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_3.csv",a3,delimiter=",")
a4 = numpy.asarray(tiempo4)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_4.csv",a4,delimiter=",")
a5 = numpy.asarray(tiempo5)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_5.csv",a5,delimiter=",")
a6 = numpy.asarray(tiempo6)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_6.csv",a6,delimiter=",")
a7 = numpy.asarray(tiempo7)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/grafica_7.csv",a7,delimiter=",")
a8 = numpy.asarray(tiempo8)
```

```
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/grafica_8.csv",a8,delimiter=",")
a9 = numpy.asarray(tiempo9)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/grafica_9.csv",a9,delimiter=",")
a10 = numpy.asarray(tiempo10)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/grafica_10.csv",a10,delimiter=",")
a11 = numpy.asarray(tiempo11)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/grafica_11.csv",a11,delimiter=",")
b1 = numpy.asarray(array_mediana)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/medianas.csv",b1,delimiter=",")
b2 = numpy.asarray(array_media)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/medias.csv",b2,delimiter=",")
b3 = numpy.asarray(array_desviacion)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/desviacion.csv",b3,delimiter=",")
b4 = numpy.asarray(array_varianza)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-analytics/csv/varianza.csv",b4,delimiter=",")
```

# ANEXO III

## Script II de validación

A continuación se muestras el script creado para la seguda fase de la validación.

En el se puede observar el uso de la extensión Python para la conexión http, la cual toma los campos “header” y “paidload” para simular las peticiones Ajax que se realizan en la interfaz web de la aplicación y poder así medir los tiempos de respuesta.

```
import http.client
import time
import statistics
import numpy

conn = http.client.HTTPConnection("localhost:8000")

simulaciones = 1001

array_media = []
array_mediana = []
array_desviacion = []
array_varianza = []
start = time.time()

# Grafica 1 [Tipos de paquete]:

tiempo1 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\npacket_types\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"type\":\"ALL\",\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-
31T23:59\",\"scenario\":\"prueba\"}\r\n-----011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "1cabc2f8-3e8d-fdae-b5b1-a7e281e365e7"
          }

        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)

        res = conn.getresponse()
        tiempo1.append(time.time()-start_time)
        data = res.read()

mediana = round(statistics.median(tiempo1),5)
media = round(statistics.mean(tiempo1),5)
desviacion = round(numpy.std(tiempo1),5)
varianza = round(statistics.variance(tiempo1),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n----------> Grafica 1 <----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# Grafica 2 [UP vs DW]:

tiempo2 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\nup_do\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
```

```
name=\"json_options\"\r\n\r\n{\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-31T23:59\",\"scenario\":\"prueba\"}\r\n-----
011000010111000001101001--"

            headers = {
              'content-type': "multipart/form-data; boundary=---011000010111000001101001",
              'cache-control': "no-cache",
              'postman-token': "c18f9c96-61c2-18bd-924b-967e767c3ca6"
              }

            start_time = time.time()
            conn.request("POST", "/ajax/", payload, headers)

            res = conn.getresponse()
            tiempo2.append(time.time()-start_time)
            data = res.read()

mediana = round(statistics.median(tiempo2),5)
media = round(statistics.mean(tiempo2),5)
desviacion = round(numpy.std(tiempo2),5)
varianza = round(statistics.variance(tiempo2),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 2 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# Grafica 3 [Tipos de paquete UP]:

tiempo3 = []
for i in range(1,simulaciones):

            payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\npacket_types\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"type\":\"UP\",\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-
31T23:59\",\"scenario\":\"prueba\"}\r\n-----011000010111000001101001--"

            headers = {
              'content-type': "multipart/form-data; boundary=---011000010111000001101001",
              'cache-control': "no-cache",
              'postman-token': "a3ab210f-18df-575c-422a-b086d6dc0598"
              }

            start_time = time.time()
            conn.request("POST", "/ajax/", payload, headers)

            res = conn.getresponse()
            tiempo3.append(time.time()-start_time)
            data = res.read()

mediana = round(statistics.median(tiempo3),5)
media = round(statistics.mean(tiempo3),5)
desviacion = round(numpy.std(tiempo3),5)
varianza = round(statistics.variance(tiempo3),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)
```

```
print ('\n------------> Grafica 3 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# Grafica 4 [Tipos de paquete DW]:

tiempo4 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\npacket_types\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"type\":\"DO\",\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-
31T23:59\",\"scenario\":\"prueba\"}\r\n-----011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "fb17cbbf-b30d-51fa-d0c6-6e3d10fa8c2a"
          }

        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)

        res = conn.getresponse()
        tiempo4.append(time.time()-start_time)
        data = res.read()

mediana = round(statistics.median(tiempo4),5)
media = round(statistics.mean(tiempo4),5)
desviacion = round(numpy.std(tiempo4),5)
varianza = round(statistics.variance(tiempo4),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 4 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# Grafica 5 [Instantes con mayor numero de desregistros]:

tiempo5 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n3\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\ndisconnections_by_time\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-31T23:59\",\"scenario\":\"prueba\"}\r\n-----
011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "b64b7f3c-250d-2872-4ab5-5384c98c3efa"
          }
```

```
            start_time = time.time()
            conn.request("POST", "/ajax/", payload, headers)

            res = conn.getresponse()
            tiempo5.append(time.time()-start_time)
            data = res.read()

mediana = round(statistics.median(tiempo5),5)
media = round(statistics.mean(tiempo5),5)
desviacion = round(numpy.std(tiempo5),5)
varianza = round(statistics.variance(tiempo5),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 5 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')

# Grafica 6 [Switches ordenados por numero de desregistros de sus hijos]:

tiempo6 = []
for i in range(1,simulaciones):

            payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n4\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\ndisconnections_switches\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-31T23:59\",\"scenario\":\"prueba\"}\r\n-----
011000010111000001101001--"

            headers = {
              'content-type': "multipart/form-data; boundary=---011000010111000001101001",
              'cache-control': "no-cache",
              'postman-token': "1ef26003-ae5f-05e9-d5d1-1274c288d72d"
              }

            start_time = time.time()
            conn.request("POST", "/ajax/", payload, headers)

            res = conn.getresponse()
            tiempo6.append(time.time()-start_time)
            data = res.read()

mediana = round(statistics.median(tiempo6),5)
media = round(statistics.mean(tiempo6),5)
desviacion = round(numpy.std(tiempo6),5)
varianza = round(statistics.variance(tiempo6),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 6 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')
```

# Grafica 7 [Nodos ordenados por numero de desregistros]:

```
tiempo7 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n4\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\ndisconnections_nodes\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-31T23:59\",\"scenario\":\"prueba\"}\r\n-----
011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "e4079f27-5c01-d0e8-bf8b-e512278b2bfc"
          }

        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)

        res = conn.getresponse()
        tiempo7.append(time.time()-start_time)
        data = res.read()

mediana = round(statistics.median(tiempo7),5)
media = round(statistics.mean(tiempo7),5)
desviacion = round(numpy.std(tiempo7),5)
varianza = round(statistics.variance(tiempo7),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n----------> Grafica 7 <----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')
```

# Grafica 8 [% de conexiones a switches]:

```
tiempo8 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n3\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\nswitches\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"mac\":\"00:80:E1:18:65:A1\",\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-
31T23:59\",\"scenario\":\"prueba\"}\r\n-----011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "16be6961-b502-5d1c-f4cc-0d410def4c44"
          }

        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)

        res = conn.getresponse()
        tiempo8.append(time.time()-start_time)
        data = res.read()
```

```
mediana = round(statistics.median(tiempo8),5)
media = round(statistics.mean(tiempo8),5)
desviacion = round(numpy.std(tiempo8),5)
varianza = round(statistics.variance(tiempo8),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 8 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')



# Grafica 9 [SNR vs Tiempo]:


tiempo9 = []
for i in range(1,simulaciones):


        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n2\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n1\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\nmac_vs_snr\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"mac\":\"00:80:e1:18:65:a1\",\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-
31T23:59\",\"scenario\":\"prueba\"}\r\n-----011000010111000001101001--"


        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "1a07d534-9495-f0c8-0c4f-0310e1e90be5"
          }


        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)


        res = conn.getresponse()
        tiempo9.append(time.time()-start_time)
        data = res.read()

mediana = round(statistics.median(tiempo9),5)
media = round(statistics.mean(tiempo9),5)
desviacion = round(numpy.std(tiempo9),5)
varianza = round(statistics.variance(tiempo9),5)


array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n------------> Grafica 9 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')



# Grafica 10 [SNR medio para cada nodo]:


tiempo10 = []
for i in range(1,simulaciones):
```

```
        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n5\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\navg_snrs\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-31T23:59\",\"scenario\":\"prueba\"}\r\n-----
011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "0b891703-9b78-98fc-a920-a022c7acf22b"
          }

        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)

        res = conn.getresponse()
        tiempo10.append(time.time()-start_time)
        data = res.read()

mediana = round(statistics.median(tiempo10),5)
media = round(statistics.mean(tiempo10),5)
desviacion = round(numpy.std(tiempo10),5)
varianza = round(statistics.variance(tiempo10),5)

array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)

print ('\n------------> Grafica 10 <-----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


# Grafica 11 [PNA count]:

tiempo11 = []
for i in range(1,simulaciones):

        payload = "-----011000010111000001101001\r\nContent-Disposition: form-data; name=\"action\"\r\n\r\ncalculate\r\n-----
011000010111000001101001\r\nContent-Disposition: form-data; name=\"row\"\r\n\r\n6\r\n-----011000010111000001101001\r\nContent-
Disposition: form-data; name=\"column\"\r\n\r\n0\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"stat_type\"\r\n\r\npna_count\r\n-----011000010111000001101001\r\nContent-Disposition: form-data;
name=\"json_options\"\r\n\r\n{\"start_time\":\"1970-01-01T00:00\",\"end_time\":\"2100-12-31T23:59\",\"scenario\":\"prueba\"}\r\n-----
011000010111000001101001--"

        headers = {
          'content-type': "multipart/form-data; boundary=---011000010111000001101001",
          'cache-control': "no-cache",
          'postman-token': "10086eec-84b6-0aaf-72d1-1b2fe050e7c5"
          }

        start_time = time.time()
        conn.request("POST", "/ajax/", payload, headers)

        res = conn.getresponse()
        tiempo11.append(time.time()-start_time)
        data = res.read()

mediana = round(statistics.median(tiempo11),5)
media = round(statistics.mean(tiempo11),5)
desviacion = round(numpy.std(tiempo11),5)
varianza = round(statistics.variance(tiempo11),5)
```

```
array_media.append(media)
array_mediana.append(mediana)
array_desviacion.append(desviacion)
array_varianza.append(varianza)


print ('\n----------> Grafica 11 <----------')
print ('Media: \t\t',media,'\t seg')
print ('Mediana: \t',mediana,'\t seg')
print ('Desviacion: \t',desviacion,'\t seg')
print ('Varianza: \t',varianza,'\t seg')


end = time.time() - start
print('\nTiempo total: \t', round(end,3),'\t seg')


a1 = numpy.asarray(tiempo1)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_1_c.csv",a1,delimiter=",")
a2 = numpy.asarray(tiempo2)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_2_c.csv",a2,delimiter=",")
a3 = numpy.asarray(tiempo3)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_3_c.csv",a3,delimiter=",")
a4 = numpy.asarray(tiempo4)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_4_c.csv",a4,delimiter=",")
a5 = numpy.asarray(tiempo5)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_5_c.csv",a5,delimiter=",")
a6 = numpy.asarray(tiempo6)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_6_c.csv",a6,delimiter=",")
a7 = numpy.asarray(tiempo7)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_7_c.csv",a7,delimiter=",")
a8 = numpy.asarray(tiempo8)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_8_c.csv",a8,delimiter=",")
a9 = numpy.asarray(tiempo9)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_9_c.csv",a9,delimiter=",")
a10 = numpy.asarray(tiempo10)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_10_c.csv",a10,delimiter=",")
a11 = numpy.asarray(tiempo11)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/grafica_ajax_11_c.csv",a11,delimiter=",")


b1 = numpy.asarray(array_mediana)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/medianas_ajax_c.csv",b1,delimiter=",")
b2 = numpy.asarray(array_media)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/medias_ajax_c.csv",b2,delimiter=",")
b3 = numpy.asarray(array_desviacion)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/desviacion_ajax_c.csv",b3,delimiter=",")
b4 = numpy.asarray(array_varianza)
numpy.savetxt("/home/mario/Documentos/PrimeAnalytics/Analizador 100% Distribuido 3.0/prime-
analytics/csv/ajax/completo/varianza_ajax_c.csv",b4,delimiter=",")
```

# ANEXO IV

## Script scale.py

A continuación se muestras el script creado para el correcto escalado y creación del Cluster.

En el se hace uso de extensiones Python para lanzar determinados comandos al terminal y obtener la información sobre las IPs necesarias y la agregación del Script sequences.sql a los Workers para conseguir el correcto funcionamiento del clúster.

```
import psycopg2
import subprocess
import os
import time


# Elegimos el numero de workers que queremos tener
n_worker = 4


for i in range(1,n_worker+1):
        scale = "docker-compose scale worker='%s'"%(i)
        os.system(scale)
        time.sleep(1)



print ('----------> Escalado a ',n_worker,' Workers')


worker_command = []
command = "docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' primeanalytics_worker_"


for i in range(1,n_worker+1):
        command2 = command + "%s"%(i)
        worker_command.append(command2)


ip = []


for index in range (0,n_worker):
        aux = subprocess.check_output(worker_command[index], shell=True)
        aux = aux[:-1]
        ip.append(aux)


for i in ip:
        database = psycopg2.connect(database="primeanalytics",user="primeanalytics",password="1234",host=i,port="5432")
        cursor = database.cursor()
        cursor.execute(open("sequences.sql","r").read())
        database.commit()
        print ('Secuencias agregadas a la IP ',i)
        time.sleep(2)


print ('----------> Completado')
```